

Prepare for the new 2009 Linux+ exam

As the Linux server and desktop markets continue to grow, so does the need for qualified Linux administrators. CompTIA's new Linux+ (Exam XK0-003) includes the very latest enhancements to the popular open source operating system. This detailed guide not only covers all key exam topics—including installation and configuration, system maintenance and operations, application and services, networking and security—it also builds your practical Linux skills with real-world examples. Inside, you'll find:

- Full coverage of all exam objectives** in a systematic approach, so you can be confident you're getting the instruction you need for the exam
- Real-world scenarios** that put what you've learned into practical context
- Challenging review questions** in each chapter to prepare you for exam day
- Exam Essentials**, a key feature in each chapter that identifies critical areas you must become proficient in before taking the exam
- A handy tear card** that maps every official exam objective to the corresponding chapter in the book, so you can track your exam prep objective by objective

Look inside for complete coverage of all exam objectives.

www.sybex.com

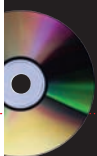
ABOUT THE AUTHOR


Roderick W. Smith, Linux+, LPIC-1, is a Linux consultant and author. His areas of expertise include Linux networking, filesystems, and cross-platform configuration. He has written over 20 books on open source technologies, including the *LPIC-1: Linux Professional Institute Certification Study Guide, 2nd Edition* and *Linux Administrator Street Smarts*, both from Sybex.


Sybex®
An Imprint of
WILEY

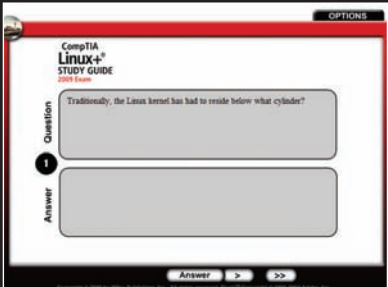


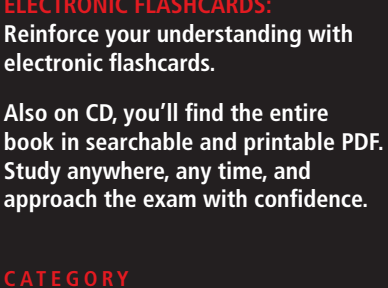
\$49.99 US
\$59.99 CN

**FEATURED ON THE CD**

**STUDY GUIDE**

**CompTIA
Linux+
STUDY GUIDE**


**SYBEX TEST ENGINE:**
Test your knowledge with advanced testing software. Includes all chapter review questions and bonus exams. Runs on both Windows and Linux.


**ELECTRONIC FLASHCARDS:**
Reinforce your understanding with electronic flashcards.


Also on CD, you'll find the entire book in searchable and printable PDF. Study anywhere, any time, and approach the exam with confidence.

CATEGORY
COMPUTERS/Certification Guides

Smith

**ISBN: 978-0-470-50384-3**

**Exam XK0-003**

**SYBEX**

Covers All Exam Objectives



Includes Real-World Scenarios and Leading-Edge Exam Prep Software Featuring:

- Linux-Compatible Custom Test Engine
- Hundreds of Sample Questions
- Electronic Flashcards
- Entire Book in PDF



CompTIA Linux+ STUDY GUIDE

Exam XK0-003

Roderick W. Smith



SERIOUS SKILLS.

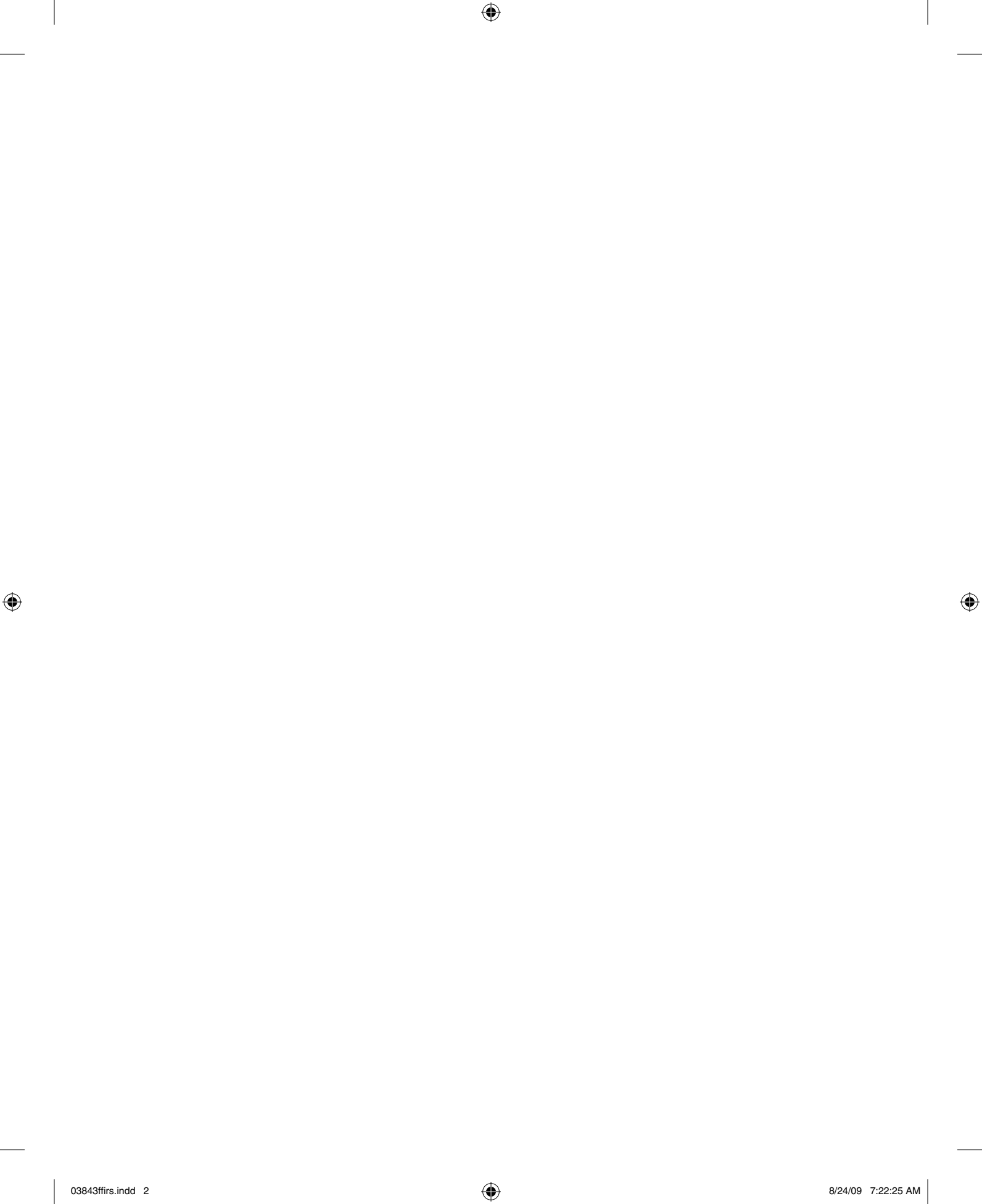


CompTIA

Linux+™

Study Guide





CompTIA

Linux+™

Study Guide



Roderick W. Smith



WILEY

Wiley Publishing, Inc.

Acquisitions Editor: Jeff Kellum
Development Editor: Stephanie Barton
Technical Editors: Elizabeth Zinkann, Beau Sanders
Production Editor: Elizabeth Britten
Copy Editor: Kim Wimpsett
Editorial Manager: Pete Gaughan
Production Manager: Tim Tate
Vice President and Executive Group Publisher: Richard Swadley
Vice President and Publisher: Neil Edde
Media Associate Project Manager: Laura Moss-Hollister
Media Associate Producers: Marilyn Hummel, Shawn Patrick
Media Quality Assurance: Josh Frank
Book Designers: Judy Fung, Bill Gibson
Compositor: Craig Woods, Happenstance Type-O-Rama
Proofreader: Jen Larsen, Word One New York
Indexer: Ted Laux
Project Coordinator, Cover: Lynsey Stanford
Cover Designer: Ryan Sneed

Copyright © 2009 by Wiley Publishing, Inc., Indianapolis, Indiana

Published simultaneously in Canada

ISBN: 978-0-470-50384-3

No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, except as permitted under Sections 107 or 108 of the 1976 United States Copyright Act, without either the prior written permission of the Publisher, or authorization through payment of the appropriate per-copy fee to the Copyright Clearance Center, 222 Rosewood Drive, Danvers, MA 01923, (978) 750-8400, fax (978) 646-8600. Requests to the Publisher for permission should be addressed to the Permissions Department, John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030, (201) 748-6011, fax (201) 748-6008, or online at <http://www.wiley.com/go/permissions>.

Limit of Liability/Disclaimer of Warranty: The publisher and the author make no representations or warranties with respect to the accuracy or completeness of the contents of this work and specifically disclaim all warranties, including without limitation warranties of fitness for a particular purpose. No warranty may be created or extended by sales or promotional materials. The advice and strategies contained herein may not be suitable for every situation. This work is sold with the understanding that the publisher is not engaged in rendering legal, accounting, or other professional services. If professional assistance is required, the services of a competent professional person should be sought. Neither the publisher nor the author shall be liable for damages arising herefrom. The fact that an organization or Web site is referred to in this work as a citation and/or a potential source of further information does not mean that the author or the publisher endorses the information the organization or Web site may provide or recommendations it may make. Further, readers should be aware that Internet Web sites listed in this work may have changed or disappeared between when this work was written and when it is read.

For general information on our other products and services or to obtain technical support, please contact our Customer Care Department within the U.S. at (877) 762-2974, outside the U.S. at (317) 572-3993 or fax (317) 572-4002.

Wiley also publishes its books in a variety of electronic formats. Some content that appears in print may not be available in electronic books.

Library of Congress Cataloging-in-Publication Data:

Smith, Roderick W.

CompTIA Linux+ study guide / Roderick W. Smith. — 1st ed.

p. cm.

ISBN 978-0-470-50384-3 (paper/cd-rom)

1. Electronic data processing personnel—Certification. 2. Operating systems (Computers)—Examinations—Study guides. 3. Computer networks—Examinations—Study guides. 4. Linux. I. Title.

QA76.3.S4765 2009

005.4'32—dc22

2009027779

TRADEMARKS: Wiley, the Wiley logo, and the Sybex logo are trademarks or registered trademarks of John Wiley & Sons, Inc. and/or its affiliates, in the United States and other countries, and may not be used without written permission. CompTIA Linux+ is a trademark of the Computing Technology Industry Association. All other trademarks are the property of their respective owners. Wiley Publishing, Inc., is not associated with any product or vendor mentioned in this book.

10 9 8 7 6 5 4 3 2 1

The logo of the CompTIA Authorized Quality Curriculum (CAQC) program and the status of this or other training material as “Authorized” under the CompTIA Authorized Quality Curriculum program signifies that, in CompTIA’s opinion, such training material covers the content of the CompTIA’s related certification exam. CompTIA has not reviewed or approved the accuracy of the contents of this training material and specifically disclaims any warranties of merchantability or fitness for a particular purpose. CompTIA makes no guarantee concerning the success of persons using any such “Authorized” or other training material in order to prepare for any CompTIA certification exam.

The contents of this training material were created for the CompTIA Linux+ exam covering CompTIA certification objectives that were current as of 2009.

How to become CompTIA certified:

This training material can help you prepare for and pass a related CompTIA certification exam or exams. In order to achieve CompTIA certification, you must register for and pass a CompTIA certification exam or exams.

In order to become CompTIA certified, you must:

1. Select a certification exam provider.
2. Register for and schedule a time to take the CompTIA certification exam(s) at a convenient location.
3. Read and sign the Candidate Agreement, which will be presented as the time of the exam(s).
4. Take and pass the CompTIA certification exam(s).

For more information about CompTIA’s certifications, such as its industry acceptance, benefits or program news, please visit <http://www.comptia.org/certifications>.

CompTIA is a not-for-profit trade information technology (IT) trade association. CompTIA’s certifications are designed by subject matter experts from across the IT industry. Each CompTIA certification is vendor-neutral, covers multiple technologies, and requires demonstration of skills and knowledge widely sought after by the IT industry.

To contact CompTIA with any questions or comments, please call 630-678-8300 or email question@comptia.org.

Dear Reader,

Thank you for choosing *CompTIA Linux+™ Study Guide*. This book is part of a family of premium-quality Sybex books, all of which are written by outstanding authors who combine practical experience with a gift for teaching.

Sybex was founded in 1976. More than 30 years later, we're still committed to producing consistently exceptional books. With each of our titles, we're working hard to set a new standard for the industry. From the paper we print on to the authors we work with, our goal is to bring you the best books available.

I hope you see all that reflected in these pages. I'd be very interested to hear your comments and get your feedback on how we're doing. Feel free to let me know what you think about this or any other Sybex book by sending me an e-mail at nedde@wiley.com. If you think you've found a technical error in this book, please visit <http://sybex.custhelp.com>. Customer feedback is critical to our efforts at Sybex.

Best regards,

A handwritten signature in black ink, appearing to read 'Neil Edde', written in a cursive style.

Neil Edde
Vice President and Publisher
Sybex, an Imprint of Wiley

From one writer to another: Lola, keep writing your stories!

Acknowledgments

Although one person's name is most prominent on the cover, this book, like most, involves many people's input. At every point along the way from project beginning to finished product, many people other than the author have had their influence. Jeff Kellum, acquisitions editor, helped set the book on course at its inception. Stephanie Barton, development editor, guided the book's development throughout the process. Liz Britten, as production editor, coordinated the work of the many others who contributed their thoughts to the book. Copy editor Kim Wimpsett helped keep my grammar and spelling on track. Elizabeth Zinkann, the technical editor, scrutinized the text for technical errors and made sure it was complete. I'd also like to thank Neil Salkind and others at Studio B, who helped get the project off the ground.

Contents at a Glance

<i>Introduction</i>	<i>xix</i>
<i>Assessment Test</i>	<i>xxix</i>
Chapter 1	Getting Started with Linux 1
Chapter 2	Using Text-Mode Commands 47
Chapter 3	Managing Processes and Editing Files 107
Chapter 4	Managing System Services 147
Chapter 5	Managing Users 187
Chapter 6	Managing Disks 237
Chapter 7	Managing Packages and System Backups 293
Chapter 8	Configuring Basic Networking 349
Chapter 9	Configuring Advanced Networking 391
Chapter 10	Configuring Network Servers I 437
Chapter 11	Configuring Network Servers II 473
Chapter 12	Securing Linux 513
Appendix	About the Companion CD 555
Glossary	559
<i>Index</i>	<i>597</i>



Contents

Introduction *xix*

Assessment Test *xxix*

Chapter 1	Getting Started with Linux	1
	Selecting an Installation Method	2
	Choosing a Distribution	2
	Selecting an Installation Medium	3
	Interacting with the Installer	5
	Performing the Installation	5
	Configuring Boot Loaders	7
	The Role of the Boot Loader	7
	Available Boot Loaders	8
	Configuring GRUB	9
	Troubleshooting Boot Problems	14
	Setting Kernel Options in GRUB	14
	Using Rescue Discs	15
	Resetting the <i>root</i> Password	16
	Examining Boot Messages with <i>dmesg</i>	16
	Configuring X	17
	Selecting an X Server	17
	Setting Up X	20
	Managing GUI Logins	25
	Using Window Managers and Desktop Environments	29
	Using Terminal Programs	29
	Managing Hardware	30
	Finding Compatible Hardware	30
	Identifying Hardware in Linux	31
	Managing Kernel Modules	35
	Summary	38
	Exam Essentials	38
	Review Questions	40
	Answers to Review Questions	44
Chapter 2	Using Text-Mode Commands	47
	Using a Command Shell	48
	Starting a Shell	49
	Using Virtual Terminals	50
	Launching Programs	50
	Using Shell Shortcuts	51
	Using the Shell's History	52

Manipulating Files and Directories	53
Navigating the Linux Filesystem	54
Manipulating Files	58
Manipulating Directories	62
Locating Files	63
Examining Files' Contents	66
Using Redirection and Pipes	69
Generating Command Lines	70
Using Device Files	71
Using File Permissions	72
Understanding Accounts and Ownership	72
Using File Access Permissions	72
Changing File Ownership and Permissions	78
Setting Default Permissions	82
Using ACLs	83
Setting Environment Variables	84
Where to Set Environment Variables	85
The Meanings of Common Environment Variables	86
Using Shell Scripts	89
Getting Help	91
Using Man Pages	91
Using Info Pages	94
Using Miscellaneous Program Documentation	95
Using Internet-Based Help Resources	96
Summary	97
Exam Essentials	98
Review Questions	99
Answers to Review Questions	103

Chapter 3 Managing Processes and Editing Files 107

Managing Processes	108
Understanding Processes	108
Examining Process Lists with <i>ps</i>	109
Restricting Processes' CPU Use	116
Killing Processes	117
Controlling Foreground and Background Processes	119
Monitoring System Statistics	119
Setting Process Permissions	123
Understanding the Risks of SUID and SGID Programs	123
Knowing When to Use SUID or SGID	123
Finding SUID or SGID Programs	124
Running Jobs at Specific Times	125
Understanding the Role of Cron	125
Creating System Cron Jobs	126
Creating User Cron Jobs	127
Using <i>at</i>	128

	Getting and Setting Kernel Information	129
	Obtaining Kernel Version Information	129
	Setting System Control Data	131
	Editing Files with Vi	133
	Using Vi Modes	134
	Editing Text	134
	Saving Changes	137
	Summary	138
	Exam Essentials	138
	Review Questions	139
	Answers to Review Questions	143
Chapter 4	Managing System Services	147
	Starting and Stopping Services	148
	Methods of Starting and Stopping Services	149
	Starting and Stopping via SysV Scripts	149
	Using Super Servers	153
	Using Custom Startup Files	158
	Setting the Runlevel	159
	Understanding the Role of the Runlevel	159
	Using <i>init</i> or <i>telinit</i> to Change the Runlevel	160
	Permanently Changing the Runlevel	162
	Configuring Log Files	163
	Understanding <i>syslogd</i>	163
	Setting Logging Options	164
	Rotating Log Files	166
	Using a Remote Server for Log Files	169
	Using Log Files	170
	Which Log Files Are Important?	170
	Using Log Files to Identify Problems	171
	Using Tools to Help Scan Log Files	172
	Summary	178
	Exam Essentials	178
	Review Questions	180
	Answers to Review Questions	184
Chapter 5	Managing Users	187
	Understanding Multiuser Concepts	188
	User Accounts: The Core of a Multiuser System	188
	Linking Users Together for Productivity via Groups	193
	Mapping UIDs and GIDs to Users and Groups	194
	Understanding Home Directories	196
	Configuring User Accounts	197
	Adding Users	197
	Modifying User Accounts	200

Deleting Accounts	207
Verifying Account Use	208
Configuring Groups	212
Adding Groups	212
Modifying Group Information	213
Deleting Groups	216
Using Common User and Group Strategies	216
Using User Private Groups	217
Using Project Groups	217
Assigning Users to Multiple Groups	218
Improving Account Security	219
Enforcing User Password Security	219
Steps for Reducing the Risk of Compromised Passwords	221
Disabling Unused Accounts	222
Using Shadow Passwords	223
Controlling System Access	224
Accessing Common Servers	224
Controlling <i>root</i> Access	225
Summary	227
Exam Essentials	227
Review Questions	229
Answers to Review Questions	233

Chapter 6 Managing Disks 237

Storage Hardware Identification	238
Types of Storage Devices	239
Linux Storage Hardware Configuration	240
Planning Disk Partitioning	242
Understanding Partitioning Systems	242
Linux Partition Requirements	244
Common Optional Partitions	245
Linux Filesystem Options	248
Partitioning Tools	250
Partition Management and Maintenance	251
Creating Partitions	251
Creating New Filesystems	256
Checking a Filesystem for Errors	258
Adding Swap Space	259
Setting Filesystem Quotas	263
Partition Control	265
Identifying Partitions	265
Mounting and Unmounting Partitions	266
Using Network Filesystems	271
Using <i>df</i>	273

	Defining Standard Filesystems	274
	Using RAID	276
	Using LVM	280
	Summary	282
	Exam Essentials	283
	Review Questions	285
	Answers to Review Questions	289
Chapter 7	Managing Packages and System Backups	293
	Understanding Package Concepts	294
	File Collections	294
	The Installed File Database	295
	Using Network Repositories	296
	Rebuilding Packages	297
	Installing and Removing Packages	298
	Handling RPM Packages	298
	Handling Debian Packages	307
	Handling Tarballs	314
	Compiling Source Code	318
	Managing Package Dependencies and Conflicts	322
	Real and Imagined Package Dependency Problems	322
	Workarounds to Package Dependency Problems	323
	Backing Up and Restoring a Computer	324
	Common Backup Hardware	324
	Common Backup Programs	326
	Performing Network Backups with <i>rsync</i>	333
	Planning a Backup Schedule	334
	Preparing for Disaster: Backup Recovery	335
	Writing to Optical Discs	336
	Linux Optical Disc Tools	336
	A Linux Optical Disc Example	337
	Creating Cross-Platform Discs	338
	Summary	339
	Exam Essentials	340
	Review Questions	341
	Answers to Review Questions	345
Chapter 8	Configuring Basic Networking	349
	Understanding Networks	350
	Basic Functions of Network Hardware	350
	Types of Network Hardware	351
	Network Packets	353
	Network Protocol Stacks	354

Network Addressing	358
Types of Network Addresses	359
Resolving Hostnames	363
Network Ports	364
Basic Network Configuration	365
Network Hardware Configuration	366
Setting Wireless Options	366
DHCP Configuration	368
Static IP Address Configuration	369
Using GUI Configuration Tools	373
Diagnosing Network Problems	375
Examining the ARP Cache	376
Testing Basic Connectivity	377
Tracing a Route	377
Checking Network Status	378
Name Server Troubleshooting	379
Using General Network Tools	380
Summary	382
Exam Essentials	382
Review Questions	384
Answers to Review Questions	388

Chapter 9	Configuring Advanced Networking	391
	Routing Between Networks	392
	Firewall Configuration	393
	Where a Firewall Fits in a Network	394
	Linux Firewall Software	395
	Common Server Ports	396
	Using <i>iptables</i>	398
	Managing Remote Logins	405
	Setting Up a Remote Access Server	405
	Using Text-Mode Logins	406
	Generating SSH Keys	408
	Using X Programs Remotely	409
	Remote GUI Logins	412
	Configuring Basic Printing	414
	The Linux Printing Architecture	414
	Understanding PostScript and Ghostscript	415
	Running a Printing System	416
	Configuring CUPS	417
	Printing to Network Printers	423
	Monitoring and Controlling Print Queues	424
	Summary	428
	Exam Essentials	428
	Review Questions	430
	Answers to Review Questions	434

Chapter 10	Configuring Network Servers I	437
	Delivering Network Information	438
	Delivering IP Addresses with DHCP	438
	Delivering Hostnames with DNS	441
	Delivering the Time with NTP	448
	Authenticating Users on the Network	454
	Using E-mail	455
	Understanding E-mail Protocols	456
	Configuring SMTP Servers	457
	Using Aliases and Forwarding E-mail	460
	Choosing a POP or IMAP Server	461
	Summary	462
	Exam Essentials	463
	Review Questions	464
	Answers to Review Questions	469
Chapter 11	Configuring Network Servers II	473
	Delivering Files Over the Network	474
	Delivering Files with Samba	474
	Delivering Files with NFS	480
	Delivering Files with FTP	483
	Configuring Web Servers	489
	Using Windows Remote Access Tools	496
	Using <i>rdesktop</i>	496
	Using VNC	497
	Deploying MySQL	499
	Picking a SQL Package	499
	Using MySQL	500
	Summary	503
	Exam Essentials	503
	Review Questions	505
	Answers to Review Questions	509
Chapter 12	Securing Linux	513
	Sources of Security Vulnerability	514
	Physical Access Problems	515
	Stolen Passwords	516
	Local Program Bugs	516
	Server Bugs	517
	Denial-of-Service Attacks	518
	Encryption Issues	518
	The Human Element	519
	Authenticating Users	520
	Understanding How Linux Authenticates Users	520
	Configuring PAM	521

Using Network Authentication	523
Using Two-Factor Authentication	526
Using GNU Privacy Guard (GPG)	526
Generating and Importing Keys	527
Encrypting and Decrypting Data	528
Signing Messages and Verifying Signatures	529
SELinux	529
Principles of SELinux	529
Configuring SELinux Running Modes	530
Security Auditing	531
Checking for Open Ports	531
Reviewing Accounts	535
Verifying Installed Files and Packages	537
Intrusion Detection	537
Symptoms of Intrusion	537
Using Snort	538
Using PortSentry	540
Using Wireshark	541
Using Tripwire	542
Generating Checksums Manually	543
Using Package Manager Checksums	544
Using <i>chkrootkit</i>	545
Monitoring Log Files	545
Summary	546
Exam Essentials	547
Review Questions	548
Answers to Review Questions	552

Appendix About the Companion CD 555

What You'll Find on the CD	556
Sybex Test Engine	556
PDF of the Book	556
Adobe Reader	556
Electronic Flashcards	557
System Requirements	557
Using the CD	557
Troubleshooting	558
Customer Care	558

Glossary 559

<i>Index</i>	597
--------------	-----

Introduction

Why should you learn about Linux? It's a fast-growing operating system, and it is inexpensive and flexible. Linux is also a major player in the small and midsize server field, and it's an increasingly viable platform for workstation and desktop use as well. By understanding Linux, you'll increase your standing in the job market. Even if you already know the Windows or Mac operating system and your employer uses these systems exclusively, understanding Linux will give you an edge when you are looking for a new job or when you are looking for a promotion. Knowing Linux will also help you to make an informed decision about if and when you should deploy Linux.

The Computing Technology Industry Association (CompTIA) has developed its Linux+ exam as an introductory certification for people who want to enter careers involving Linux. The exam is meant to certify that an individual has the skills necessary to install, operate, and troubleshoot a Linux system and is familiar with Linux-specific concepts and basic hardware.

The purpose of this book is to help you pass the 2009 version of the Linux+ exam (XK0-003). Because this exam covers basic Linux installation, configuration, maintenance, applications, networking, and security, those are the topics that are emphasized in this book. You'll learn enough to get a Linux system up and running and how to configure it for many common tasks. Even after you've taken and passed the Linux+ exam, this book should remain a useful reference.



NOTE

The original Linux+ exam was released in 2001, but in the fast-changing world of computers, updates became desirable within a few years. Thus, CompTIA released an updated version of the Linux+ exam in 2005 and again in 2009. This book covers this latest Linux+ exam, rather than earlier versions. Earlier editions of this book covered the earlier Linux+ exams.

What Is Linux?

Linux is a clone of the Unix OS that has been popular in academia and many business environments for years. Formerly used exclusively on large mainframes, Unix and Linux can now run on small computers—which are actually far more powerful than the mainframes of just a few years ago. Because of its mainframe heritage, Unix (and hence also Linux) scales well to perform today's demanding scientific, engineering, and network server tasks.

Linux consists of a kernel, which is the core control software, and many libraries and utilities that rely on the kernel to provide features with which users interact. The OS is available in many different distributions, which are bundlings of a specific kernel with specific support programs. These concepts are covered at greater length in Chapter 1.

Why Become Linux+ Certified?

Several good reasons to get your Linux+ certification exist. The CompTIA Candidates Information packet lists five major benefits:

Provides proof of professional achievement Certifications are quickly becoming status symbols in the computer service industry. Organizations, including members of the computer service industry, are recognizing the benefits of certification, such as Linux+. Organizations are pushing for their members to become certified. Every day, more people are putting the CompTIA official certification logo on their business cards.

Increases your marketability Linux+ certification makes individuals more marketable to potential employers. Also, Linux+ certified employees might receive a higher salary base because employers won't have to spend as much money on vendor-specific training.

Provides an opportunity for advancement Most raises and advancements are based on performance. Linux+ certified employees work faster and more efficiently. The more productive employees are, the more money they will make for their company; and, of course, the more money they make for the company, the more valuable they will be to the company. So, if employees are Linux+ certified, their chances of getting promoted will be greater.

Fulfills training requirements Each year, more and more major computer hardware vendors, including (but not limited to) IBM, Hewlett-Packard, and Novell, are recognizing CompTIA's certifications as prerequisites in their own respective certification programs. The use of outside certifications like Linux+ has the side benefit of reducing training costs for employers. Because more and more small companies are deploying the flexible and inexpensive OS we call Linux, the demand for experienced users is growing. CompTIA anticipates that the Linux+ exam, like the A+ exam, will find itself integrated into various certification programs as well.

Raises customer confidence As the IT community, users, small business owners, and the like become more familiar with the Linux+ certified professional moniker, more of them will realize that the Linux+ professional is more qualified to work in their Linux environment than is a noncertified individual.

How to Become Linux+ Certified

The Linux+ certification is available to anyone who passes the test. You don't have to work for a particular company. It's not a secret society. It is, however, an elite group.

The exam is administered by Thomson Prometric and Pearson VUE. The exam can be taken at any Thomson Prometric or Pearson VUE testing center. If you pass, you will get a certificate in the mail from CompTIA saying that you have passed, and you will also receive a lapel pin and business cards. To find the Thomson Prometric testing center nearest you, call (800) 755-EXAM (755-3926). Contact (877) 551-PLUS (551-7587) for Pearson VUE information.

To register for the exam with Thomson Prometric, call (800) 776-MICRO (776-4276), or register online at <http://securereg3.prometric.com>. To register with Pearson VUE, call (877) 551-PLUS (551-7587), or register online at <http://www.vue.com/comptia/>. However you do it, you'll be asked for your name, mailing address, phone number, employer, when and where you want to take the test (i.e., which testing center), and your credit card number (arrangement for payment must be made at the time of registration).

Who Should Buy This Book

Anybody who wants to pass the Linux+ exam may benefit from this book. If you're new to Linux, this book covers the material you will need to learn the OS from the beginning, and it continues to provide the knowledge you need up to a proficiency level sufficient to pass the Linux+ exam. You can pick up this book and learn from it even if you've never used Linux before, although you'll find it an easier read if you've used Linux at least casually for a few days. If you're already familiar with Linux, this book can serve as a review and as a refresher course for information with which you might not be completely familiar. In either case, reading this book will help you pass the Linux+ exam.

This book is written with the assumption that you know at least a little bit about Linux (what it is and possibly a few Linux commands). This book also assumes that you know some basics about computers in general, such as how to use a keyboard, how to insert a CD-ROM or DVD-ROM into an optical drive, and so on. Chances are you have used computers in a substantial way in the past—perhaps even Linux, as an ordinary user, or maybe you have used Windows or Mac OS. This book does *not* assume that you have extensive knowledge of Linux system administration, but if you've done some system administration, you can still use this book to fill in gaps in your knowledge.

How This Book Is Organized

This book consists of 12 chapters plus supplementary information, including a glossary, this introduction, and the assessment test after the introduction. The chapters are organized as follows:

- Chapter 1, “Getting Started with Linux,” covers issues you should consider before you install Linux on a computer, as well as low-level hardware, boot, and X (GUI) configuration. Because there are so many Linux distributions, each of which has its own installation method, neither this book nor the Linux+ exam covers Linux installation in great detail.
- Chapter 2, “Using Text-Mode Commands,” provides a grounding in using Linux at the command line. The chapter begins with a look at command shells generally and moves on to commands used to manipulate files. The chapter also describes environment variables and introduces the basics of creating shell scripts, which can help automate otherwise tedious tasks.

- Chapter 3, “Managing Processes and Editing Files,” begins with information on Linux’s tools for managing processes—that is, running programs. You’ll learn how to terminate misbehaving programs and run jobs at scheduled times, among other things. This chapter concludes with a look at editing text files in Linux.
- Chapter 4, “Managing System Services,” covers methods used to start, stop, and otherwise control the many programs that run constantly on a Linux system. These programs include servers and nonserver programs that run in the background to provide necessary Linux features. This chapter also covers log files, which record information from system services, such as login attempts and the loading of kernel drivers.
- Chapter 5, “Managing Users,” describes how to create and maintain user accounts; it also covers some basic user-related security issues. Because Linux is a clone of Unix, it includes extensive support for multiple users, and understanding Linux’s model for user accounts is critical to many aspects of Linux’s operation.
- Chapter 6, “Managing Disks,” covers Linux’s approach to hard disks, partitions, and the filesystems they contain. Specific topics include how to create and manage filesystems, how to create and use a RAID array, and how to create and use LVM volumes.
- Chapter 7, “Managing Packages and System Backups,” describes Linux’s tools for maintaining installed software. This chapter covers common package management tools, procedures for compiling software from source code, and tools for backing up and restoring that software (and user files) for security in case of disk errors or other problems.
- Chapter 8, “Configuring Basic Networking,” covers how to use Linux on a network. This chapter includes an overview of what a network is, including the popular TCP/IP networking tools on which the Internet is built. The subject of network diagnostics is also covered.
- Chapter 9, “Configuring Advanced Networking,” describes more advanced network topics. These include router configuration, firewalls, remote logins, and printer configuration. (Linux’s printing tools are inherently network-enabled.)
- Chapter 10, “Configuring Network Servers I,” is the first of two chapters devoted to network servers. This chapter covers Dynamic Host Configuration Protocol (DHCP), Domain Name System (DNS), Network Time Protocol (NTP), and e-mail servers.
- Chapter 11, “Configuring Network Servers II,” continues the look at network servers. It covers file servers (Samba, the Network File System [NFS], the File Transfer Protocol [FTP], and the Apache Web server), Windows remote-access tools, and the Structured Query Language (SQL) database server.
- Chapter 12, “Securing Linux,” covers the important topic of keeping your system secure. Specific topics covered here include authentication, personal encryption tools, SELinux, security auditing, and intrusion detection.

Each chapter begins with a list of the CompTIA Linux+ objectives that are covered in that chapter. (The book doesn’t cover objectives in the same order as CompTIA lists them,

and a few numbered objectives are split across multiple chapters, so don't be alarmed when you notice gaps or repeats in the sequence.) At the end of each chapter, you'll find two sections you can use to help prepare for the exam:

Exam Essentials This section summarizes important information that was covered in the chapter. You should be able to perform each of the tasks or convey the information requested.

Review Questions Each chapter concludes with 20 review questions. You should answer these questions and check your answer against the one provided after the questions. If you can't answer at least 80 percent of these questions correctly, go back and review the chapter, or at least those sections that seem to be giving you difficulty.



The review questions, assessment test, and other testing elements included in this book are *not* derived from the CompTIA Linux+ exam questions, so don't memorize the answers to these questions and assume that doing this will enable you to pass the Linux+ exam. You should learn the underlying topic, as described in the text of the book. This will let you answer the questions provided with this book *and* pass the exam. Learning the underlying topic is also the approach that will serve you best in the workplace—the ultimate goal of a certification such as Linux+.

To get the most out of this book, you should read each chapter from start to finish and then check your memory and understanding with the chapter-end elements. Even if you're already familiar with a topic, you should skim the chapter; Linux is complex enough that there are often multiple ways to accomplish a task, so you may learn something even if you're already competent in an area.

Bonus CD-ROM Contents

This book comes with a CD-ROM that contains several additional elements. Items available on the CD-ROM include the following:

Book contents as a PDF file The entire book is available as a fully searchable PDF that can be read on most OSs, including Linux and Windows.

Electronic “flashcards” The CD-ROM includes 150 questions in “flashcard” format (a question followed by a single correct answer). You can use these to review your knowledge of the Linux+ exam objectives.

Sample tests All of the questions in this book appear on the CD-ROM—both the 30-question assessment test at the end of this introduction and the 240 questions that consist of the twelve 20-question “Review Questions” sections for each chapter. In addition, there are two 65-question bonus exams.

Conventions Used in This Book

This book uses certain typographic styles in order to help you quickly identify important information and avoid confusion over the meaning of words such as on-screen prompts. In particular:

- *Italicized text* indicates key terms that are described at length for the first time in a chapter. (Italics are also used for emphasis.)
- A **monospaced font** indicates the contents of configuration files, messages displayed at a text-mode Linux shell prompt, filenames, text-mode command names, and Internet URLs.
- *Italicized monospaced text* indicates a variable—information that differs from one system or command run to another, such as the name of a client computer or a process ID number.
- **Bold monospaced text** is information that you're to type into the computer, usually at a Linux shell prompt. This text can also be italicized to indicate that you should substitute an appropriate value for your system. (When isolated on their own lines, commands are preceded by nonbold monospaced \$ or # command prompts.)

In addition to these text conventions, which can apply to individual words or entire paragraphs, a few conventions highlight segments of text.



A note indicates information that's useful or interesting but that's somewhat peripheral to the main text. A note might be relevant to a small number of networks, for instance, or it may refer to an outdated feature.



A tip provides information that can save you time or frustration and that may not be entirely obvious. A tip might describe how to get around a limitation or how to use a feature to perform an unusual task.



Warnings describe potential pitfalls or dangers. If you fail to heed a warning, you may end up spending a lot of time recovering from a bug, or you may even end up restoring your entire system from scratch.

Sidebar

A sidebar is like a note but longer. The information in a sidebar is useful, but it doesn't fit into the main flow of the text.



Real World Scenario

Example

A real world scenario is a sidebar that provides information that's particularly grounded in the real world. Such text might be based on my own experiences or provide practical information that might not be obvious from reading the basic facts about a topic.

The Exam Objectives

Behind every computer industry exam you can be sure to find exam objectives—the broad topics in which exam developers want to ensure your competency. The official CompTIA objectives for the Linux+ exam are listed here. (They're also printed at the start of the chapters in which they're covered.)



Exam objectives are subject to change at any time without prior notice and at CompTIA's sole discretion. Please visit the Linux+ Certification page of CompTIA's Web site (<http://www.comptia.org/certifications/listed/linux.aspx>) for the most current listing of exam objectives.

Domain 1.0 Installation and Configuration

- 1.1 Compare and contrast installation sources (Physical installation media: CD-ROM, DVD; Network types: HTTP, FTP, NFS).
- 1.2 Implement partitioning schemes and filesystem layout using the following tools and practices (LVM, RAID, fdisk, parted, mkfs).
- 1.3 Explain the purpose for using each of the following filesystem types (Local: EXT2, EXT3, Reiser, FAT, NTFS, VFAT, ISO9660. Network: NFS, SMBFS/CIFS).
- 1.4 Conduct routine mount and unmount of filesystems (mount, umount, /etc/fstab).
- 1.5 Explain the advantages of having a separate partition or volume for any of the following directories (/boot, /home, /tmp, /usr, /var, /opt).
- 1.6 Explain the purpose of the following directories (/ , /bin, /dev, /etc, /mnt, /proc, /root, /sbin, /user/bin, /usr/local, /usr/lib, /usr/lib64, /usr/share, /var/log).
- 1.7 Configure the boot process including the following (GRUB: /boot/grub/grub.conf, /boot/grub/menu.lst, grub-install, grub).

1.8 Perform the following package management functions (Install, remove and update programs: rpm [rpm -Uvh, rpm -qa, rpm -e, yum], deb [dpkg -i, dpkg -r, apt-get, apt-cache search], source [./configure, make, make install, make uninstall, tar, make clean, autoconf, make test, tar.gz, INSTALL, bzip, gzip]; Resolve dependencies; Add and remove repositories).

1.9 Configure profile and environment variables system-wide and at the user level (PS1, PS2, PATH, EDITOR, TERM, PAGER, HOME, PRINTER).

1.10 Troubleshoot boot issues using the following tools (kernel options, single user mode [including recovering the root user], Rescue—live CDs, DVDs, and USB keys, dmesg).

1.11 Manage devices using the following tools (lsusb, lspci, lsmod, /sys, modprobe, /proc, /etc/modules.conf, /etc/modprobe.conf, Hardware Compatibility List [HCL]).

Domain 2.0 System Maintenance and Operations

2.1 Given a scenario, use the following fundamental Linux tools, techniques, and resources (Directory navigation: cd, ls, pushd, popd, pwd; File commands: file, test, find, locate, slocate, which, whereis, ln, ls -F, mknod, touch, mkdir, mv, cp, rm, cd; file types [hard links, soft links, directory, device file, regular file, named pipe]; File editing with Vi; Process management: ps, kill, top, iostat, pstree, nice, renice, signals, PID, PPID; I/O redirection: <, >, =, ==, |, ;, tee, xargs, STDIN, STDOUT, STDERR; Special devices: /dev/null, /dev/random, /dev/zero, /dev/urandom; System documentation: Man pages [man#, apropos, makewhatis, whatis], Info pages, /usr/share/docs]; Virtual consoles; Kernel/architecture information: cat, /proc/version, uname, common sysctl settings, /etc/sysctl.conf).

2.2 Conduct basic tasks using BASH (Basics of scripting [only: execute permission, #!/bin/bash, sh script]; Shell features: history, tab completion).

2.3 Given a scenario, analyze system and application logs to troubleshoot Linux systems (Common log files: /var/log/messages, /var/log/syslog, /var/log/maillog, /var/log/secure, /var/log/lastlog; Rotated logs; Searching and interpreting log files: grep, tail -f, awk, sed).

2.4 Conduct and manage backup and restore operations (Copying data: rsync and ftp; Archive and restore commands: cpio, tar, dump, restore, dd; Write to removable media: CD-RW, DVD-RW).

2.5 Explain the following features and concepts of X11 (Starting and stopping X11, Differences between the X11 client and server, Window managers and display managers [KDM, GDM], Multiple desktops, X11 configuration file [xorg.conf], Terminal emulators [xterm, etc.]).

2.6 Explain the difference in runlevels and their purpose (Command: init; Runlevels: 0 – Halt, 1 – single-user mode, 2 – single-user mode with networking, 3 – networked multi-user mode, 4 – user configurable, 5 – X11 multi-user mode, 6 – reboot).

2.7 Manage filesystems using the following (Check disk usage: `df`, `du`; Quotas: `edquota`, `repquota`, `quotacheck`; Check and repair filesystems: `fsck`; Loopback devices: ISO filesystems; NFS: configuration, `mount`, `exportfs`, `fstab`, `/etc/exports`, `showmount`; Swap: `mkswap`, `swapon`, `swapoff`).

2.8 Implement task scheduling using the following tools (cron: `cron.allow`, `cron.deny`, `crontab` command syntax, `crontab` file format, `at`: `atq`).

2.9 Utilize performance monitoring tools and concepts to identify common problems (Commands: `sar`, `iostat`, `vmstat`, `uptime`, `top`. Load average).

Domain 3.0 Application and Services

3.1 Manage Linux system services using the following (`/etc/init.d`: `start`, `stop`, `restart`; `inetd`; `xinetd`; `chkconfig`).

3.2 Implement interoperability with Windows using the following (`rdesktop`—client, `vnc`—server and client, Samba—server and client [`smb.conf`, `winbind`, `lmhosts`]; Security and authentication [Kerberos]).

3.3 Implement, configure and maintain Web and FTP services (Apache: maintain PHP settings [`php.ini`]; edit Apache configuration files : enable and disable modules; containers: virtual hosts, directories; access control : [`.htaccess`]; CGI: `ExecCGI`, `ScriptAlias`; commands: `apachectl [-t, -S, graceful, restart]`; configuring apache logs, FTP services: configure FTP users [`/etc/ftpusers`, `chroot`]; configure anonymous access).

3.4 Given a scenario, explain the purpose of the following web-related services (Tomcat, Apache, Squid).

3.5 Troubleshoot web-related services using the following utilities (Commands: `curl`, `wget`, `ftp`, `telnet`).

3.6 Given a scenario, troubleshoot common FTP problems (Active vs. passive, ASCII vs. binary).

3.7 Given a scenario, perform the following MySQL administrative tasks (Locate configuration file, Starting and stopping, Test the configuration).

3.8 Explain the purpose of each of the following mail services, protocols, and features (Protocols: SMTP, IMAP, POP3; MTA: Postfix, Sendmail; Email aliases: `/etc/aliases`, `newaliases`).

3.9 Deploy and manage CUPS print services (enable and disable queues, Web management interface [port 631], Printing commands: `lp`, `lpq`, `lpstat`, `cancel`).

3.10 Set up, install, configure, and maintain a BIND DNS server and related services (DNS utilities: `named`, `rndc`; Config file locations [`/var/named`]; Forward zones, reverse zones, root hints).

3.11 Perform basic administration of the DHCP server (/etc/dhcpd.conf, dhcpd.leases).

3.12 Given a scenario, troubleshoot NTP related issues (/etc/ntp.conf, ntpdate, date, ntpq -p).

Domain 4.0 Networking

4.1 Identify common networking ports and the associated service (20, 21, 22, 23, 25, 53, 80, 110, 123, 143, 443, 631, 3306, /etc/services).

4.2 Execute network interface configuration using the following (dhclient, dhcpd, ifconfig, iwconfig, route, ifup, ifdown, network configuration files).

4.3 Implement configurations and/or configuration changes for the following (Packet filtering: iptables; Hostname lookup: /etc/hosts, /etc/nsswitch.conf, /etc/resolv.conf).

4.4 Explain the different DNS record types and the process of DNS resolution (Local resolution, TTL/caching, Root name servers, A, MX, PTR, CNAME, NS, TXT).

4.5 Troubleshoot basic connectivity issues using the following tools (netstat, ping, traceroute, arp, telnet, route).

4.6 Troubleshoot name resolution issues using the following tools (dig, host, nslookup, hostname).

Domain 5.0 Security

5.1 Manage and monitor user and group accounts using the following (Tools: useradd, userdel, usermod, groupadd, groupdel, groupmod, lock, who, w, last, whoami. Files: /etc/skel, /etc/passwd, /etc/shadow, /etc/group).

5.2 Given a scenario, select the appropriate file permissions and ownership and troubleshoot common problems (Tools: chmod, chown, chroot, chgrp, lsattr, chattr, umask. Special permissions: setuid, setgid, sticky bit).

5.3 Explain the basics of SELinux (Running modes [Enabled, Disabled, Permissive]).

5.4 Given a scenario, implement privilege escalation using the following (sudo, su).

5.5 Explain the appropriate use of the following security related utilities (nmap, Wireshark, NESSUS, Snort, Tripwire).

5.6 Use checksum and file verification utilities (md5sum, sha1sum, gpg).

5.7 Deploy remote access facilities using the following (SSH: Secure tunnels, SFTP, X11 forwarding, Keygen; VNC).

5.8 Explain the methods of authentication (PAM, LDAP, NIS, RADIUS, Two-factor authentication).

Assessment Test

1. Where may GRUB be installed?
 - A. The MBR, a Linux partition's boot sector, or a floppy disk
 - B. The MBR, a Linux partition's boot sector, or a Windows partition's boot sector
 - C. A Linux partition's boot sector or a Windows partition's boot sector
 - D. The MBR, a floppy disk, or a swap partition
2. Which of the following tools is it *most* important to have available on an emergency recovery disk?
 - A. fdformat
 - B. OpenOffice.org
 - C. mkfs
 - D. traceroute
3. Which of the following types of information are you likely to see in log files? (Choose all that apply.)
 - A. Information about a user launching a text editor to edit a file in the user's directory
 - B. Successful uses of the `su` command to acquire root privileges
 - C. Failed attempts to log in to a server controlled through `xinetd`
 - D. Failed attempts by a user to read another user's files via the `cat` command
4. What does the `-t` parameter to `telinit` control?
 - A. The time between a polite shutdown of unneeded servers (via `SIGTERM`) and a forceful shutdown (via `SIGKILL`)
 - B. The time between issuing the `telinit` command and the time the runlevel change takes place
 - C. The runlevel that's to be entered on completion of the command
 - D. The message sent to users before the runlevel change is enacted
5. You type `ifconfig eth0` and see the following line in the output. What does this output mean?
Link encap:Ethernet HWaddr 00:A0:CC:24:BA:02
 - A. The Ethernet card is malfunctioning, with an error code of 00:A0:CC:24:BA:02.
 - B. The Ethernet card has a hardware address of 00:A0:CC:24:BA:02.
 - C. The hardware is incapable of establishing an Ethernet link with other devices.
 - D. The 00 at the start of the HWaddr indicates that the interface is not yet active.

6. Which of the following is it wise to do when deleting an account with `userdel`?
 - A. Ensure that the user's password isn't duplicated in `/etc/passwd` or `/etc/shadow`.
 - B. Search the computer for stray files owned by the former user.
 - C. Change permissions on system files to prevent the user from accessing them remotely.
 - D. Delete the user's files with a utility that overwrites former file contents with random data.
7. An `ls -l` command reveals that the `loud` file has a permission string of `crw-rw----` and ownership by the user `root` and group `audio`. Which of the following is a true statement about this file?
 - A. Only `root` and the account that created it may read or write the file.
 - B. The file is a directory, as indicated by the leading `c`.
 - C. Anybody in the `audio` group may read from and write to the file.
 - D. The command `chmod 660 loud` will make it accessible to more users.
8. Which of the following is commonly found in `/etc/inetd.conf` entries for servers but not in the equivalent entries in `/etc/xinetd.conf` or a file in `/etc/xinetd.d`?
 - A. A call to `tcpd`
 - B. A specification of the protocol, such as `tcp`
 - C. A specification of the user, such as `nobody`
 - D. Arguments to be passed to the target server
9. Under which of the following circumstances will a `chmod` command not work?
 - A. The user issuing the command doesn't own the file but does own and have write permission to the directory in which the file resides.
 - B. The `root` user issues the command on a file that resides in a read/write filesystem, although the file itself has no write permissions active.
 - C. The owner of the file issues the command, but the file's permissions don't grant the owner write access to the file.
 - D. The owner of the file issues the command, but the file resides in a directory to which the owner has read but not write access.
10. What is the function of the `su` command?
 - A. It gives the named user superuser privileges.
 - B. It acquires the named user's privileges, or the superuser's if no username is specified.
 - C. It acquires superuser privileges for the user who runs the command.
 - D. It gives everybody currently logged in superuser privileges.

11. Under which of the following circumstances would you be *most* likely to need to edit the `/etc/fstab` file's contents?
 - A. You've added a hard disk to the computer.
 - B. You've inserted a Zip disk into the computer's Zip drive.
 - C. The system performs a disk check and reports no errors.
 - D. You've changed `/boot/grub/menu.lst` boot options.
12. Which category of attack denies you the use of your equipment?
 - A. DoS
 - B. Core
 - C. Rooted
 - D. Phish
13. You want to set up a firewall on a Linux computer. Which of the following tools might you use to accomplish this task?
 - A. Apache
 - B. iptables
 - C. wall
 - D. TCP Wrappers
14. Which of the following is the intended purpose of the `rc.local` or `boot.local` startup script?
 - A. It sets the system's time zone and language defaults.
 - B. It holds startup commands created for its specific computer.
 - C. It displays startup messages to aid in debugging.
 - D. It verifies that all other startup scripts are operating correctly.
15. What role does BIND fill?
 - A. It allows programs to use dynamic library files.
 - B. It translates between hostnames and IP addresses.
 - C. It ties together POP and SMTP servers.
 - D. It binds an IP address to a network interface.
16. You suspect that a router between you and `ftp.abigisp.net` is malfunctioning and preventing a connection. What tool might you use to track down which system is the problem?
 - A. nslookup
 - B. ping
 - C. traceroute
 - D. netstat

17. Which of the following commands is *most* likely to stop a runaway process with PID 2939?
- A. `kill -s SIGHUP 2939`
 - B. `kill -s SIGTERM 2939`
 - C. `kill -s SIGKILL 2939`
 - D. `kill -s SIGDIE 2939`
18. Which of the following is *not* one of the responsibilities of CUPS?
- A. Maintaining the printer queues
 - B. Accepting print jobs from remote systems
 - C. Managing the scanner hardware on combination scanner/printers
 - D. Sending data to printers
19. Which of the following commands displays the contents of a tarball, including file sizes and time stamps?
- A. `tar xzf theprogram-1.2.3.tgz`
 - B. `tar tzf theprogram-1.2.3.tgz`
 - C. `tar tvzf theprogram-1.2.3.tgz`
 - D. `tar x theprogram-1.2.3.tgz`
20. Which of the following commands replaces jpertwee's current cron job with my-cron?
- A. `crontab -r my-cron`
 - B. `crontab -e my-cron`
 - C. `crontab jpertwee my-cron`
 - D. `crontab -u jpertwee my-cron`
21. How would you direct the output of the `uptime` command to a file called `uptime-stats.txt`?
- A. `echo uptime uptime-stats.txt`
 - B. `uptime > uptime-stats.txt`
 - C. `uptime | uptime-stats.txt`
 - D. `uptime < uptime-stats.txt`
22. A workstation ordinarily runs with a load average of 0.25. Suddenly, its load average is 1.25. Which of the following might you suspect, given this information? (Choose all that apply.)
- A. The workstation's user may be running more programs or more CPU-intensive programs than usual.
 - B. A process may have hung—locked itself in a loop consuming CPU time but doing no useful work.
 - C. A process may have begun consuming an inordinate amount of memory.
 - D. The CPU may be malfunctioning and require replacement.

23. You've installed Linux on a new computer, but you've been unable to get X to function in anything better than 640 × 480 (VGA) video mode. After you investigate, you learn that the video card's chipset is unsupported by the standard drivers in XFree86 3.3.6, XFree86 4.x, or X.org-X11. Which of the following is not an option you might consider to resolve the situation?
- A. Replace the video card.
 - B. Obtain an XFree86 4.x driver from the card's manufacturer.
 - C. Replace your distribution's X server with Accelerated-X.
 - D. Upgrade the card's memory to allow higher video resolutions.
24. The final step of your company's procedures for creating a new server requires you to store information on /dev/hda's MBR partition table in an ASCII file named `documentation.txt`. Which of the following commands will allow you to accomplish this action?
- A. `df /dev/hda > documentation.txt`
 - B. `parted -l /dev/hda > documentation.txt`
 - C. `fdisk -l /dev/hda > documentation.txt`
 - D. `du /dev/hda > documentation.txt`
25. You are configuring your company firewall and have been told that TCP and UDP data to port 53 must be allowed through. By default, what server uses this port?
- A. NNTP
 - B. PortMapper
 - C. NetBIOS
 - D. BIND
26. Your company makes computer hardware, and you want to enable customers to download Linux, Windows, and MacOS drivers and documentation files for this hardware. Which of the following are the *best* choices to enable users to retrieve such files? (Choose two.)
- A. SMTP
 - B. FTP
 - C. HTTP
 - D. NFS
27. Which of the following statements are true of SSH? (Choose all that apply)
- A. Most default configurations allow `root` to log in directly.
 - B. Encryption makes SSH safer than Telnet.
 - C. The default port used is 53.
 - D. By default, SSH uses UDP.

28. Where may the Linux root (/) partition reside?
- A. On a primary or logical partition
 - B. On a logical partition only
 - C. On a primary partition only
 - D. On a partition that falls below the 1,024th cylinder
29. What tool can diagnose and fix many common Linux filesystem problems?
- A. mkfs
 - B. fsck
 - C. chkdsk
 - D. scandisk
30. Which of the following tools enables you to digitally “sign” e-mail messages as a way of proving to recipients that you were the true sender?
- A. GPG
 - B. SSH
 - C. BIND
 - D. Winbind

Answers to Assessment Test

1. A. GRUB may reside in any of the locations listed in option A. If you install it in a FAT or NTFS partition (used by DOS or Windows), these partitions will be damaged, and if you install GRUB in a swap partition that is then used, GRUB will be wiped out. See Chapter 1 for more information.
2. C. Option C, `mkfs`, is a tool for creating a new filesystem, which is something you're likely to need to do in an emergency recovery situation. The first option, `fdformat`, does a low-level format on a floppy disk, OpenOffice.org is an office productivity suite, and `traceroute` helps diagnose network connectivity problems. You're unlikely to need to use any of these tools from an emergency disk. See Chapter 1 for more information.
3. B, C. Log files record messages generated by the kernel, servers, and certain security-related system utilities. The `su` command typically records a summary of its actions in log files, and `xinetd` typically records login failures, although these defaults can be changed. Text editors seldom log information, nor do simple file-viewing utilities, even if they're asked to violate file security measures. See Chapter 4 for more information.
4. A. When shutting down certain servers, `telinit` first tries asking them to shut themselves down by sending a `SIGTERM` signal. The server can then close open files and perform other necessary shutdown housekeeping. If the servers don't respond to this signal, `telinit` becomes more forceful and passes a `SIGKILL` signal, which is more likely to work but doesn't give the server a chance to shut itself down in an orderly fashion. The `-t` parameter specifies the time between these two signals. See Chapter 4 for more information.
5. B. The `Link encap: Ethernet` portion of the output indicates that the device uses Ethernet to establish links with other systems. `HWaddr 00:A0:CC:24:BA:02` specifies the hardware address (aka the Media Access Control, or MAC, address). Ethernet cards use these addresses to "talk" to each other; Linux's TCP/IP stack locates a MAC address when given a TCP/IP address, if the target system is on the same network segment. (If not, Linux locates the gateway system and passes data through it.) See Chapter 8 for more information.
6. B. Tracking down and removing or changing the permissions of a former user's files can prevent confusion or possibly even spurious accusations of wrongdoing in the future. Unless the user was involved in system cracking, there's no reason to think that the user's password would be duplicated in the password database. No system file's ownership or permissions should need changing when deleting a user. Although overwriting deleted files with random data may be useful in some high-security environments or with unusually sensitive data, it's not a necessary practice on most systems. See Chapter 5 for more information.
7. C. The second set of permission bits (`rw-`) indicates that the file's group (`audio`) may read from and write to the file. This permission string ensures that, if `audio` has more than one member, multiple users may access the file. The leading `c` indicates that the file is a character device file, not a directory. The command `chmod 660 loud` will not change the file's permissions; `660` is equivalent to `rw-rw----`. See Chapter 2 for more information.

8. A. The TCP Wrappers program, `tcpd`, includes security features that are largely provided directly by `xinetd`, so most systems that use `xinetd` don't call `tcpd` from `xinetd`. The other options appear in both types of files, although arguments for the server aren't required for either super server. See Chapter 4 for more information.
9. A. Only the file's owner and `root` may change permissions on a file via `chmod`. Whether the file is writeable by the owner is irrelevant, as is whether the directory in which the file resides is writeable. See Chapter 2 for more information.
10. B. Typing **`su username`** gives the person who runs the command the privileges associated with that username, assuming that the person who runs the command successfully enters the user's password. When the username isn't specified, `root` is assumed. The `su` command also runs a program as the specified user. Normally, this is a shell, but you can specify another program using a command-line argument. Although option C describes part of what `su` can do, option C is incomplete; option B is a more complete answer. The `su` command does not give superuser privileges to the named user, nor does it give everybody who's logged in superuser privileges. See Chapter 5 for more information.
11. A. To use disk space on a new computer, you must add one or more `/etc/fstab` entries for the new disk partitions (unless you intend to manually mount the new disk's partitions as `root` whenever you need them). A Zip disk is just like any other removable medium; `/etc/fstab` can contain an entry for the drive, which doesn't need changing to read a new disk. Disk checks don't change partition tables, and so they don't require `/etc/fstab` to be edited. The `/boot/grub/menu.lst` boot configuration won't alter partition layouts either. See Chapter 6 for more information.
12. A. A denial of service (DoS) attack is any type of attack that denies you the use of your equipment. Core is not a type of attack. Rooted is a term used to describe `root` access being obtained by a hacker, while phishing involves sending bogus e-mails or setting up fake Web sites that lure unsuspecting individuals into divulging sensitive financial or other information. See Chapter 12 for more information.
13. B. Option B, `iptables`, is a tool for configuring the 2.4.x and 2.6.x Linux kernel's firewall features. (The `ipfwadm` and `ipchains` programs perform these tasks for the 2.0.x and 2.2.x kernels, respectively.) Apache is a Web server, and `wall` sends messages to all currently logged-on users. TCP Wrappers controls access to specific servers, but it isn't a firewall per se. See Chapter 9 for more information.
14. B. These scripts hold startup commands individualized for their host ("local") computer, as opposed to those that are provided with the distribution. In principle, these scripts *could* be used for any of the other listed purposes, but this isn't their usual function. See Chapter 4 for more information.
15. B. BIND is the Berkeley Internet Name Domain—a name server. Normally, BIND runs only on your network's name servers; other computers on your network point to a DNS server that runs BIND or similar software. See Chapter 10 for more information.
16. C. You can use `traceroute` to check the time for data transfers to, and the reliability of, every router between you and a destination system. This data should help you identify a failure point on a connection. See Chapter 8 for more information.

17. C. Many servers use `SIGHUP` as a code to reread their configuration files; this signal doesn't normally terminate the process. `SIGTERM` is a polite way to stop a process; it lets the process control its own shutdown, including closing open files. `SIGKILL` is a more forceful method of termination; it's more likely to work than `SIGTERM`, but open files won't be saved. There is no `SIGDIE` signal. See Chapter 3 for more information.
18. C. The Common Unix Printing System (CUPS) accepts print jobs from local and remote systems, maintains print queues, and sends data to printers (both local and remote). It does *not*, however, manage the scanner side of combination scanner/printer hardware; that task is handled by Scanner Access Now Easy (SANE) or other scanner software. See Chapter 9 for more information.
19. C. Option A extracts files from the archive without displaying their names. Option B lists the files in the archive, but without the `--verbose (v)` option, it doesn't list file sizes or time stamps. Option D will cause `tar` to attempt to extract the named file from its standard tape device. See Chapter 7 for more information.
20. D. The `-r` option removes a user's cron job, and the `-e` option edits the user's cron job. Any parameter following either of these is interpreted as a username, so both options A and B interpret `my-cron` as the username. Option C is malformed, but it will have the effect of installing the file `jpertwee` as the cron job for the user who types the command. The `-u jpertwee` parameter in option D correctly specifies the user as `jpertwee`, and the last parameter (`my-cron`) is the file holding the cron job specification. See Chapter 3 for more information.
21. B. The output redirection operator is `>`, so option B sends the output of `uptime` to `uptime-stats.txt`. The `echo` command displays information on the screen, so option A simply causes `uptime` `uptime-stats.txt` to appear. Option C uses a pipe. If `uptime-stats.txt` were a program, it would process the output of `uptime`, but the result of this command will probably be a `file not found` or `permission denied` error. Option D uses an *input* redirection operator, so `uptime` receives the contents of `uptime-stats.txt` as its input. See Chapter 2 for more information.
22. A, B. Sudden jumps in load average indicate that programs are making heavier demands on the CPU than is normal. This may be because of legitimate factors such as users running more programs or more demanding programs, or it could mean that a program has locked itself into an unproductive loop. Memory use isn't reflected in the load average. A malfunctioning CPU is likely to manifest itself in system crashes, not a change in the load average. See Chapter 3 for more information.
23. D. Upgrading the video memory won't overcome an unsupported chipset, although it might help if the board works at up to some resolution but no higher. Some manufacturers provide XFree86 or X.org-X11drivers on their Web sites. Replacing the video card or replacing XFree86 or X.org-X11 are also possible courses of action, but you should investigate compatibility before you make a purchase. See Chapter 1 for more information.
24. C. The command `fdisk -l /dev/hda > documentation.txt` will store information on `/dev/hda`'s partition table in the file `documentation.txt`. The other utilities listed will not show the information about the partition table that you would want to record in this file. See Chapter 6 for more information.

25. D. The Berkeley Internet Name Domain (BIND) server, which performs DNS name resolution, uses port 53 by default. The Network News Transfer Protocol (NNTP) uses port 119, PortMapper uses 111, and NetBIOS uses ports 137 through 139. See Chapter 8 for more information.
26. B, C. The question requires a publicly-accessible file download system using a widely-used network protocol. Both the File Transfer Protocol (FTP) and the Hypertext Transfer Protocol (HTTP; the Web's main protocol) fit the bill, although an FTP server will need to be configured for anonymous access. You might want to run option A's Simple Mail Transfer Protocol (SMTP) to receive e-mail from customers, but it won't help to achieve the stated goals. The Network File System (NFS) is a file-sharing protocol that could, in principle, be used to achieve the question's goals; however, Windows-using clients are unlikely to be able to easily access an NFS server and most Web browsers don't support NFS, making it a more awkward choice in this role even for Linux users. Thus, option D is a poor choice, and since the question asks for precisely two answers, B and C are both superior responses. See Chapter 11 for more information.
27. A, B. Most default SSH configurations allow `root` to log in directly. Although SSH's encryption makes this practice much safer than the equivalent when using Telnet, you can gain the added benefit of requiring two passwords by disabling direct `root` logins via SSH. The default port used by SSH is 22, and it is a TCP protocol. See Chapter 9 for more information.
28. A. Linux isn't fussy about primary versus logical partition types, even for the root partition. On old BIOSs or with old versions of LILO, the kernel must reside below the 1,024th cylinder, but this isn't a requirement of the root partition. See Chapter 6 for more information.
29. B. Option B, `fsck`, is Linux's filesystem check utility. It's similar in purpose to the DOS and Windows CHKDSK and ScanDisk utilities, but these DOS and Windows utilities don't work on Linux filesystems like `ext2fs` or `ReiserFS`. Option A, `mkfs`, creates new filesystems; it doesn't diagnose or fix filesystem problems. See Chapter 6 for more information.
30. A. The GNU Privacy Guard (GPG) package provides the feature described in the question. (It also supports encrypting files and e-mail messages.) The Secure Shell (SSH) program is a remote login tool; although it's also an encryption tool, it wasn't designed to do as the question states. The Berkeley Internet Name Domain (BIND) daemon is a DNS name server; it can't do as the question suggests. The Winbind package is a network authentication tool that permits Linux systems to use a Windows domain controller for authentication. It doesn't enable digital signing of files or e-mail messages. See Chapter 12 for more information.

Chapter 1

Getting Started with Linux

THE FOLLOWING COMPTIA OBJECTIVES ARE COVERED IN THIS CHAPTER:

- ✓ **1.1 Compare and contrast installation sources.**
- ✓ **1.7 Configure the boot process including the following (GRUB: /boot/grub/grub.conf, /boot/grub/menu.lst, grub-install, grub).**
- ✓ **1.10 Troubleshoot boot issues using the following tools (kernel options, single user mode [including recovering the root user], Rescue—live CDs, DVDs, and USB keys, dmesg).**
- ✓ **1.11 Manage devices using the following tools (lsusb, lspci, lsmod, /sys, modprobe, /proc, /etc/modules.conf, /etc/modprobe.conf, Hardware Compatibility List [HCL]).**
- ✓ **2.5 Explain the following features and concepts of X11 (Starting and stopping X11, Differences between the X11 client and server, Window managers and display managers (KDM, GDM), Multiple desktops, X11 configuration file (xorg.conf), Terminal emulators (xterm, etc.).**



Before you can begin using Linux, you must have a computer that runs the OS. Although computers pre-loaded with Linux exist, many people choose to install the OS themselves. Thus, this book begins with an examination of this topic. You should be aware, however, that Linux isn't a single OS, but rather a family of OSs, so this book doesn't provide a step-by-step description of Linux installation, but only an overview of some of the important concepts and choices available.

This chapter also covers some critical low-level Linux configuration issues. These are the boot process, the graphical user interface (GUI), and Linux hardware management tools. Booting Linux is done with a tool known as a *boot loader*, which can be configured to boot with a variety of options. Linux's GUI is known as the *X Window System* (X or X11 for short). It's different from the GUIs in Microsoft Windows or Mac OS, so it requires special attention. Managing hardware, although in some respects an advanced topic, is very fundamental to the computer's operation, so this chapter concludes with a look at this topic.

Selecting an Installation Method

If you're installing Linux yourself, you have several choices to make even before you sit down at the computer to do the job. The first of these is choosing a *distribution*. This is a collection of all the software that makes up a working computer; several different Linux distributions are available. Once you select a distribution, you may need to choose what installation method to use and how to interact with the installer. Once this is done, you can actually perform the installation.

Choosing a Distribution

Any OS consists of a series of software components that interact with one another to produce a working system. The lowest-level component is known as the *kernel*. The kernel interacts directly with the hardware and manages all the other programs. Technically, the name *Linux* applies only to the Linux kernel; everything else that makes up a working Linux system is outside of the Linux kernel, and most of these tools run on non-Linux systems, such as FreeBSD, Solaris, Mac OS, and even Windows.

The collection of a specific Linux kernel along with specific versions of other tools (servers, shells, e-mail clients, and so on) and configuration files to glue everything

together makes up a distribution. Dozens of Linux distributions exist, but the following handful are the most popular general-purpose distributions:

Debian If you want a distribution that conforms strictly to free software principles, Debian may be what you want. It uses less in the way of flashy GUI front-ends to configuration tools than do many other distributions. See <http://www.debian.org> for more information.

Fedora This distribution is a free variant of Red Hat. It's very popular among hobbyists and small organizations that don't need extensive support. See <http://fedoraproject.org> for more information.

Mandriva Two older distributions, Mandrake and Conectiva, merged to become Mandriva. This distribution, headquartered at <http://www.mandriva.com>, includes variants intended to run from USB flash drives.

Red Hat This distribution is among the oldest of the popular distributions. It is the commercial variant of Fedora, and it's a popular choice among businesses that want official support. See <http://www.redhat.com> for more information.

SUSE This distribution is a general-purpose distribution with both commercial (SUSE; see <http://www.novell.com/linux>) and fully open source (OpenSUSE; see <http://www.opensuse.org/en>) variants.

Ubuntu You can find this distribution at <http://www.ubuntu.com>. It's a variant of Debian that adds more in the way of GUI tools and other features that appeal to less geeky users. It's very popular among hobbyists and home users.

This list is by no means comprehensive; distributions such as Gentoo, Slackware, and others all appeal to certain people. Others are highly specialized—say, for scientific data collection or use on old hardware. If you need to select a Linux distribution, I recommend you peruse the Web sites for the six distributions in the preceding list to decide what to run. Factors you might consider include cost, hardware requirements, paid support, popularity (for ease of finding free support on the Internet), and what your friends and colleagues run.

Most modern distributions, including Fedora, Mandriva, Red Hat, and SUSE, use the RPM Package Manager (RPM; a recursive acronym) for software installation. Debian and Ubuntu use Debian packages for this purpose. This detail is important because it determines what software you'll use for maintaining your software—a topic that's covered in detail in Chapter 7, “Managing Packages and System Backups.” If you don't know what to select, don't worry too much about it, but be sure that any distribution you pick uses one of these two methods. Other methods, such as the Portage system used by Gentoo, may work perfectly well, but they aren't covered by the Linux+ exam. If you can install two distributions, pick one that uses RPMs and another that uses Debian packages, since the Linux+ exam covers both tools.

Selecting an Installation Medium

Linux can be booted and installed from any of several different media—floppy disks, CD-ROMs, network connections, and so on. For both booting and installing files, different media offer different advantages and disadvantages.

The Boot Method

Linux installer programs run within Linux itself. This means that in order to install Linux, you must be able to boot a small Linux system, which is provided by the distribution maintainer. This system is useful only for installing Linux and sometimes for doing emergency maintenance. It typically fits on one or two floppy disks or can boot from a bootable CD-ROM.

Modern BIOSs include options for the selection of a boot medium. Typical choices include the floppy disk, CD-ROM drive, PATA hard disk, SATA hard disk, SCSI hard disk, and USB media. In addition, some network cards include BIOSs that enable a computer to boot from files stored on a server. In theory, any of these media can be used to boot a Linux installer.

Although many boot methods are possible, the most common method by far is to use a bootable CD-ROM or DVD-ROM. If you buy a boxed Linux package, it will come with a bootable disc; or you can download an image from the distribution's Web site, burn it, and boot it.

In the past, floppy disks and booting from DOS or Windows were common methods of launching an installer. These methods have fallen by the wayside, but it's conceivable you'll still run into them, particularly on old, small, or specialized distributions.

Ultimately, the boot method is unimportant, because the same installation programs run no matter what method you choose. Pick the boot method that's most convenient for your hardware and the form of installation medium you've chosen.

Installation Media

The installation medium is the physical form of the source of the Linux files. These are the most common choices:

Optical discs If you buy Linux in a store or from an online retailer, chances are you'll get an optical disc. In fact, most distributions come on multiple CD-ROMs or a single DVD-ROM. Optical disc installations tend to be quick. Most distribution maintainers offer CD-ROM and DVD-ROM image files that you can burn to CD-Rs yourself. To find optical disc image files, check <http://iso.linuxquestions.org>, <ftp://sunsite.unc.edu/pub/linux/distributions> or your chosen distribution's Web or FTP site.

Network If you have a fast network connection, many distributions enable you to install via network connections. You must typically download a small image file for a boot CD-ROM, burn that image to disc, and boot it. This boot disc has only a minimal software set, so when you install from the network, you'll download only what you want to install. The drawback to network installations is that they tend to be slower than installs from CD-ROMs. They require more information from the user, and so they can be more difficult for a new user to get working. They can also fail midway if a network connection goes down or a server stops responding. Network installations may use any of several protocols to transfer files, including FTP, HTTP (Web), SMB (Windows file sharing), and NFS (Unix/Linux file sharing). Precisely which protocols are supported varies from one distribution to another.

Not all distributions support all of these installation options. All mainstream distributions support installation from optical discs, and most support at least one form of network installation. Beyond this, you should check the documentation for the distribution.

As a general rule of thumb, installing from DVD-ROMs makes the most sense on modern systems, since these computers invariably have DVD-ROM drives, and using DVD-ROMs means you won't be asked to swap discs mid-installation, as is likely if you use CD-ROMs. CD-ROMs are useful on older computers, though.

For network installation, FTP and HTTP are common choices for direct installation from remote servers. Both methods work well, but if your network has a firewall that requires use of a proxy server, you may need to enter extra information to have the installer use the proxy server. SMB and NFS are more commonly used on local networks. If you host a distribution's files on a local system, you could use these protocols (or FTP or HTTP) to perform network installations from your local server.

Interacting with the Installer

Most methods of Linux installation require you to make decisions during the process. You may need to tell the system how to partition your hard disk, what your network settings are, and so on. To handle such interactions, distribution maintainers have developed three methods of data entry:

GUI interactions Most Linux distributions employ a GUI installer; the system boots up into a basic GUI display, automatically configures the keyboard and mouse, and then proceeds to offer options. This method of installation is relatively comfortable to most new Linux users; however, the installer may fail to correctly identify the video display and may therefore revert to a text-mode interface.

Text-based interactions Some distributions default to a text-based installer, in which the computer asks questions that require keyboard responses. Typically, you select options from a text-based menu system, so you don't need to know Bash (described in more detail in Chapter 2, "Using Text-Mode Commands") or be otherwise familiar with Linux details. If a GUI installer behaves strangely, or if you prefer a text-based installer, you may be able to enter one even on distributions that use a GUI installer by default. Typically, the boot disc provides a menu early in the boot process with a prompt that explains how to enter the text-mode installer.

Scripted installations If you plan to install Linux many times on identical hardware, a scripted install may be just the ticket. In this type of install, you create a configuration file that describes precisely what you want to do. You then provide this file to the installer, which can do the rest of the job without pestering you with questions. Using a scripted installer requires highly distribution-specific knowledge.

Performing the Installation

Unfortunately, Linux distributions' installers vary substantially in how they work. You should consult your distribution's documentation to learn the details. Generally speaking,

the installer guides you through several steps, each of which sets options for particular OS features:

Language options You'll typically be asked to confirm your language. In fact, you may be asked to do this twice, in order to set the display language and the layout of your keyboard.

Disk partitioning Hard disks are typically split into multiple sections, or *partitions*, which hold data of different types. If you're installing Linux on a blank hard disk, you can probably get by with the default partitioning options. If you need Linux to coexist with another OS or if you have specialized needs, you may want to consult Chapter 6, "Managing Disks," before installing Linux.

Boot options You may be asked to set various boot options, such as enabling the computer to boot another OS in addition to Linux. The upcoming section "Configuring Boot Loaders" describes this topic in more detail.

Network configuration Linux installers typically enable you to set basic network options. Chapter 8, "Configuring Basic Networking," covers this topic in detail. For now, you should know that most networks employ the Dynamic Host Configuration Protocol (DHCP) to set most network options. If your network uses DHCP, setting the DHCP option should get basic network features working. If your network doesn't use DHCP, you'll need to ask your network administrator for advice. If in doubt, you can leave network configuration until later—at least, if you're installing from an optical disc or other local media.

X configuration Modern Linux distributions typically detect your video display hardware and set it up reasonably for use by the X Window System, Linux's GUI environment; however, you may want or need to fine-tune your monitor's resolution or enter other technical data. If this step gives you problems, you can put it off until later. The upcoming section "Configuring X" covers this topic in detail.

Time options You can set the current time and time zone as part of the system installation. One unusual feature of Linux relates to the choice of Universal Coordinated Time (UTC) vs. local time. Linux computes times based on UTC (which is closely related to Greenwich Mean Time, or GMT) and then converts those times to local time based on your time zone. DOS and Windows, however, use local time internally. For this reason, the hardware clock in computers is often set to local time. Linux handles daylight saving time changes more easily if you set your hardware clock to UTC. Thus, you have the option of using either approach. Generally speaking, you should use UTC if Linux is the only OS on the computer or if you only multiboot to other UTC-using OSs, such as FreeBSD or Mac OS. If your system multiboots with Windows, though, you may want to set the hardware clock to local time.

Package selection All distributions install a base set of programs (or *packages*). Some distributions give you options during installation about what additional software to install. This task can sometimes be overwhelming. If you're in doubt, leave the defaults; you can always install software later, as described in Chapter 7.

Account creation You'll usually have to set a password for *root*, which is the Linux administrative account. Most installers also give you the option of creating one or more user accounts during installation. Account management is covered in more detail in Chapter 5, "Managing Users."

These tasks may be performed in almost any order, and some distributions add more tasks or omit some of them.

Once the basic installation is done, the installer will reboot the computer. With any luck, you'll be greeted by a text-mode or GUI login prompt. You can then enter your username and password to see a working Linux system. Chapter 2 describes the commands used at a text-mode Linux shell. After a GUI login, you'll see a screen from which you can run various programs via a point-and-click interface similar to that in Windows or Mac OS.

The rest of this chapter covers various topics related to booting and hardware configuration. Some of the tasks described in these sections require you to work at a text-mode command shell. You can either log into a text-mode session or locate an option called *xterm*, *Konsole*, *shell*, *terminal*, or something similar from the GUI menu.



If X is running on your computer, you can access a full-screen text-mode session by pressing `Ctrl+Alt+Fn`, where *n* is a number, typically from 1 through 6. To switch back to X, you can press `Alt+F7` (Fedora uses `Alt+F1` for this purpose). These keystrokes switch between *virtual terminals*, which enable you to run multiple text-mode programs, each with its own display. X occupies one virtual terminal of its own.

Configuring Boot Loaders

The Linux kernel is at the heart of a Linux computer; in fact, technically speaking, the kernel *is* Linux—everything else is support programs. Because the kernel must run before Linux is completely booted, the kernel must be loaded into memory in a unique way. A program known as a *boot loader* handles this task. Several boot loaders are available, some of which can boot a Linux kernel directly, and others of which require help to do the job.



This section describes boot loaders for x86 and x86-64 systems using a legacy BIOS. If you're using Linux on another architecture, such as PowerPC or Alpha, the available boot loaders will be different. Consult your distribution's documentation for details.

The Role of the Boot Loader

When it's first powered up, an x86 CPU checks a specific area of memory for code to execute. This code is the BIOS in most systems. You're probably familiar with the BIOS through your computer's BIOS setup screens, which enable you to configure features such as RAM timing and whether or not onboard ports are active. The BIOS also provides code that allows the computer to boot. The BIOS checks the first sector of your hard disk (or of your

floppy disk, CD-ROM, or other disk devices, depending on the BIOS's capabilities and configuration) for a small boot loader program. This program normally resides on the master boot record (MBR) of a hard disk or the boot sector of a floppy disk. The MBR resides on the first sector of a hard disk and controls the boot process. A boot sector is the first sector of a floppy or of a hard disk partition and also controls the boot process. (In the case of a partition's boot sector, it's used after the MBR.)



The BIOS, as it currently exists, is extremely old and limited. A new system, known as the Extensible Firmware Interface (EFI), is poised to replace the BIOS. Intel-based Macintoshes already use EFI, as do a few other systems; however, EFI is not yet common on commodity PCs. If your system uses EFI, its boot process will differ from that described here.

In the case of a PC that runs nothing but Windows, the boot loader in the MBR is hard-coded to check for a secondary boot loader in the active primary partition—that is, a partition that's been marked as holding a bootable OS. This secondary boot loader directly loads the Windows kernel. The approach in Linux is similar, but standard Linux boot loaders are more complex. The Linux Loader (LILO) and the Grand Unified Boot-loader (GRUB) are the most common Linux boot loaders. Both programs enable you to boot the Linux kernel or to redirect the boot process to another OS.

In some cases, a system uses multiple boot loaders. One resides in the MBR, and another resides in the boot sector of an individual disk partition. (OSs on different partitions can each have their own boot sector–based boot loaders.) In this configuration, the MBR-based boot loader is the *primary boot loader*, and the one in a partition's boot sector is a *secondary boot loader*. Some boot loaders work in only one of these positions. It's often possible for a secondary boot loader to redirect the boot process to a different partition, in which case that partition's boot loader becomes the tertiary boot loader, although the configuration is the same as for secondary status.

Available Boot Loaders

Many OSs ship with their own boot loaders, and others are available from third parties. These are some of the most common boot loaders:

LILO This boot loader can directly boot a Linux kernel, and it can function as either a primary or a secondary boot loader. It may also be installed on a floppy disk. LILO can redirect the boot process to non-Linux partitions, and so it can be used to select Linux or Windows in a dual-boot system. Although once very popular, LILO has been largely eclipsed by GRUB as the boot loader of choice for Linux.

GRUB This boot loader is more or less the standard Linux boot loader. GRUB can be installed in the same locations as LILO—a floppy disk, the MBR, or the boot sector of a Linux partition. It can directly load the Linux kernel, as well as some other OS kernels, or it can redirect the boot process to another boot loader to load other OSs, such as Windows.

OS Loader This is one name by which Windows NT/200x/XP/Vista's boot loader goes. Another is NTLDR. This is a secondary boot loader that cannot directly boot Linux, but it can boot a disk file that can contain LILO or GRUB and hence boot Linux indirectly. It's common on some dual-boot installations.

LOADLIN This is an unusual boot loader in that it's neither a primary nor a secondary boot loader. Rather, it's a DOS program that can be used to boot Linux after DOS has already loaded. It's particularly useful for emergency situations because it enables you to boot a Linux kernel using a DOS boot floppy, and you can also use it to pass kernel parameters to influence the booted system's behavior. LOADLIN comes with most Linux distributions, generally in a directory on the main installation CD-ROM.

Many additional third-party boot loaders are available, most of which cannot directly boot a Linux kernel but can boot a partition on which LILO or GRUB is installed. For this reason, this chapter emphasizes configuring GRUB—this boot loader can be used to boot Linux, whether it functions as the primary, secondary, or tertiary boot loader. If you opt to use GRUB as a secondary boot loader, you'll need to consult the documentation for your primary boot loader to learn how to configure it.



On a Linux-only system, there's no need to deal with a third-party boot loader; LILO or GRUB can function as a primary boot loader without trouble on such systems. Third-party boot loaders are most useful when you have two or more OSs installed and particularly when LILO or GRUB has trouble redirecting the boot process to the other OSs, which is rare.

The usual configuration for GRUB is to place it in the MBR. Even in a Linux-only situation, however, it's sometimes desirable to place GRUB in the Linux boot partition. Used in this way, a standard DOS/Windows MBR will boot Linux *if* the Linux boot partition is a primary partition that's marked as active. This configuration can be particularly helpful in Windows/Linux dual-boot configurations because Windows tends to overwrite the MBR at installation. Therefore, putting GRUB in the Linux boot sector keeps it out of harm's way, and you can get GRUB working after installing or reinstalling DOS or Windows by using the DOS or Windows FDISK program and marking the Linux partition as active. If GRUB is on the MBR and is wiped out, you'll need to boot Linux in some other way, such as by using LOADLIN, and then rerun `grub-install` to restore GRUB to the MBR.

Configuring GRUB

GRUB is a collection of several components, including the boot loader code proper, a configuration file, and a set of utilities for installing and manipulating the boot loader code. The boot loader code can read the configuration file, so there's no need to reinstall the boot loader code whenever you change your GRUB configuration. You can even place the configuration file on a non-Linux partition, which can be handy for quickly reconfiguring GRUB from another OS.



GRUB wasn't developed exclusively for Linux. It can be installed from, and used to boot, a wide variety of OSs. Its Web page is <http://www.gnu.org/software/grub>. Most Linux distributions use GRUB Legacy (version 0.97 or earlier). GRUB 2 is currently in development.



Real World Scenario

The 1024-Cylinder Limit

One bane of the PC world that reared its ugly head twice in the 1990s was the so-called *1024-cylinder limit*. This limit is derived from the fact that the x86 BIOS uses a three-number scheme for addressing hard disk sectors. Each sector is identified by a cylinder number, a head number, and a sector number, known collectively as the sector's *CHS address*. The problem is that each of these values is limited in size. The cylinder number, in particular, is allotted only 10 bits and so cannot exceed 2^{10} , or 1,024, values. In conjunction with the limits for sectors and heads, this restricted addressable ATA hard disk size to 504MB in the early 1990s.

When disks larger than 504MB became common, BIOSs were adjusted with *CHS translation* schemes, which allowed them to juggle numbers between cylinders, heads, and sectors. This increased the limit to just under 8GB. A similar scheme abandoned CHS addressing for BIOS-to-disk communications but retained it for BIOS-to-software communications. This was known as *linear block addressing (LBA)* mode.

These limits never affected Linux once it had booted, because Linux could handle more than 10-bit cylinder values, and it could access disks directly using LBA mode. The Linux boot process was limited, however, because LILO (this was pre-GRUB) relied on CHS addressing via the BIOS to boot the kernel. Therefore, the Linux kernel has traditionally had to reside below the 1,024-cylinder mark.

Today, all new BIOSs include support for so-called *extended INT13* calls, which bypass the CHS addressing scheme. These BIOSs support booting an OS from past the 1,024-cylinder mark on a hard disk, but only if the boot loader and OS support this feature. Recent versions of LILO and GRUB support extended INT13 calls, so new Linux distributions can be installed anywhere on a hard disk—if the BIOS supports this feature.

Setting Global GRUB Options

The traditional location for the GRUB configuration file is `/boot/grub/menu.lst`. Fedora, Gentoo, and Red Hat, though, ship with a version of GRUB that uses `/boot/grub/grub.conf` as the configuration file. Whatever the name, the GRUB configuration file has the same basic form, as illustrated in Listing 1.1.

Listing 1.1: Sample menu.lst File

```

default=0
timeout=4
splashimage=(hd0,3)/grub/splash.xpm.gz
title Linux (2.6.29)
    root (hd0,3)
    kernel /bzImage-2.6.29 ro root=/dev/hda9 mem=256M
    boot
title Windows
    rootnoverify (hd0,1)
    chainloader +1
    boot

```

**NOTE**

Chapter 3, “Managing Processes and Editing Files,” describes one of many text editors available for Linux.

Because GRUB wasn’t designed exclusively for Linux, it introduces a new way of referring to hard disks and their partitions. Linux refers to hard disks using files in the `/dev` directory, such as `/dev/hda` and `/dev/hda9`. Disks may have names beginning with `hd` (for most PATA disks) or `sd` (for SCSI disks and most SATA disks). Following that string is a letter that refers to the physical disk, so a system could have, for instance, `/dev/sda` and `/dev/sdb`. Partitions on disks are given numbers starting with 1, as in `/dev/sda1` or `/dev/sdb7`.

GRUB uses strings of the form `(hdx,y)` to identify disks, where `x` is a disk number and `y` is a partition number. (The `y` and preceding comma may be omitted to refer to an entire disk or its MBR.) Both the `x` and the `y` are numbered starting from 0, which contrasts with Linux’s numbering partitions starting with 1. Thus, Linux’s `/dev/hda9` is GRUB’s `(hd0,8)`. GRUB doesn’t distinguish between PATA and SCSI disks; `hd0` is the first disk recognized by the BIOS, `hd1` is the second disk, and so on.

The first three lines of Listing 1.1 set global options:

Default OS The `default=0` line tells GRUB to boot the first OS defined in the file by default. If this line read `default=1`, the default would be the second OS, and so on.

Timeout period The `timeout=4` line sets the timeout before booting the default OS to 4 seconds.

Splash image The third line in Listing 1.1 sets a splash image—an image that’s displayed as part of the boot process. Many Linux distributions ship a splash image with their GRUB files to make for a fancier boot loader menu, but you can omit this line if you like. This example uses a GRUB-style hard disk specification to point to the image file. In this case, it’s the `grub/splash.xpm.gz` file on the fourth partition on the first disk (probably `/dev/hda4` or `/dev/sda4`). Depending on where this partition is mounted, that could be `/grub/splash.xpm.gz`, `/boot/grub/splash.xpm.gz`, or some other location.

Setting OS Boot Options

The two OS definitions in Listing 1.1 both begin with the keyword `title`, which provides a label for the OS that's displayed by GRUB when it boots. Subsequent lines may be indented to help distinguish between the OS definitions, but this indentation is optional. Important features of OS definitions include the following:

Root partition The `root` option identifies the GRUB root partition, which is the partition on which the GRUB configuration files reside. If you did *not* set aside a separate partition for `/boot` when you installed Linux, this line will identify the Linux root (`/`) partition, and subsequent file references will be relative to the Linux root partition. If you used a separate `/boot` partition, though, chances are the GRUB root partition will be the Linux `/boot` partition, and GRUB references to files in Linux's `/boot` directory will omit that directory name. Listing 1.1 identifies the GRUB root partition as `(hd0,3)`, which is `/dev/hda4` on a PATA system.



GRUB can read files from several filesystems, including ext2fs, ext3fs, ReiserFS, FAT, and FFS. You can use any of these filesystems as your GRUB root partition. If you want to use another filesystem, such as JFS or XFS, as your Linux root partition, you should split off your GRUB root partition from the Linux root partition.

Linux kernel The `kernel` option identifies a Linux kernel or a kernel for certain other Unix-like OSs, such as a GNU Hurd kernel. This reference is relative to the GRUB root partition, as defined by `root`. You can also pass kernel options on this line. Note that the `root` option passed to the Linux kernel identifies the Linux root partition using a Linux device filename, but the `root` option in the GRUB OS definition identifies the GRUB root partition. The two might be the same, but they might not be. In the case of Listing 1.1, they aren't the same—the GRUB root partition is `(hd0,3)`, or `/dev/hda4`, whereas the Linux root partition is `/dev/hda9`. Chances are `/dev/hda4` is the Linux `/boot` partition. The `ro` option passed on the `kernel` line tells the kernel to mount the root partition in read-only mode initially, just as the `read-only` line does in `lilo.conf`.

Root partition without verification The `rootnoverify` option works just like the `root` option, except that it tells GRUB it shouldn't try to access files on the partition in question. It's most often found when booting non-Linux and non-Unix OSs, such as DOS or Windows.

Chain loader The `chainloader +1` line in Listing 1.1 tells the system to load the first sector of the root partition and pass execution to it. This option is common when booting DOS, Windows, or other OSs that place boot loader code in their boot sectors.

Boot directive The `boot` line tells GRUB to actually boot the kernel or boot sector for the OS in this definition. In practice, it can often be omitted.

In order to boot, the GRUB boot loader code must reside in the MBR, the boot partition's boot sector, or a floppy disk. You can do this by using the `grub` utility, which you launch from a text-mode login or `xterm`:

```
# grub
grub> root (hd0,3)
```



```
grub> setup (hd0)
grub> quit
```

These commands set the GRUB root partition (the same as the one defined in your `menu.lst` or `grub.conf` file), install the boot loader code to the MBR of the hard disk (that is, to `hd0`), and exit from the utility. If you want to install the boot loader to a partition, you'd use **setup (hd0,3)** or some other partition identifier rather than using **setup (hd0)**. The `grub-install` program provides a simplified method of performing these steps:

```
# grub-install (hd0)
```

This command installs GRUB to the MBR of the first disk. It should be able to locate the GRUB root partition automatically.

If you installed a distribution that uses GRUB by default, you shouldn't have to perform any of these steps; GRUB should already be installed and working. You might need to reinstall GRUB from an emergency boot system if it becomes corrupted, though, and you might want to replace the installed system if you learn of a serious GRUB bug. If you just want to add a new kernel or OS to your existing GRUB installation, you do *not* need to reinstall the boot loader code; you need only edit the `menu.lst` or `grub.conf` file.

Adding a New Kernel or OS to GRUB

You can add a new kernel or OS to GRUB by copying an existing entry (or using one in Listing 1.1 as a model) and modifying it to suit your needs. When trying a new kernel, don't replace your old kernel; instead, add the new kernel to the `/boot` directory, and add a description of the new kernel to the GRUB configuration file. Remember to change the title line so that you can tell your two kernels apart. When you reboot the computer, you should be able to select the new kernel or OS from the list; there's no need to reinstall the GRUB boot loader code using the `grub` or `grub-install` tool.

Naming Kernel Files

A good practice when adding a new kernel is to give it a name that includes its version number or other identifying information. For instance, Listing 1.1's kernel is called `bzImage-2.6.29`, identifying it as a 2.6.29 kernel. If you had such a kernel and wanted to try adding a new feature (say, XFS support), you might call this new kernel `bzImage2.6.29-xfs`. There are no hard-and-fast rules for such naming, so use whatever system you like. As a general rule, though, the base of the name begins with `vmLinux` (for a "raw" kernel file), `vmLinux` (for a kernel compressed with `gzip`), `zImage` (another name for a kernel compressed with `gzip`), or `bzImage` (for a kernel compressed in a way that supports booting larger kernel images). Most distributions use `vmLinux` for their kernels, but locally compiled kernels usually go by the `bzImage` name.

Troubleshooting Boot Problems

Linux normally boots correctly immediately after installation; however, sometimes it doesn't. What's more, boot problems can develop after installation. Knowing how to handle such problems can be a necessary skill. You can interact with GRUB to enter options at boot time and use rescue discs to modify your working system. You can use several different methods to reset the `root` password, should you forget what it is—an extremely important skill, should the need arise! Finally, you can examine boot messages to help identify and debug boot problems.

Setting Kernel Options in GRUB

When GRUB loads, it normally presents a series of kernel and OS options, as defined in the `menu.lst` or `grub.conf` configuration file. You highlight the line with the title of the kernel or OS you want to boot and press the Enter key to boot.

Rather than accept the default options, though, you can edit them: instead of pressing the Enter key, you can type **e** (without pressing Enter) to edit the entry. Once you do this, GRUB presents the options from the configuration file related to the entry you've selected. You can use the keyboard's arrow keys to select a line and then type **e** again to edit that line. Alternatively, you can type **0** or **o** to create a new blank line before or after the selected one. Once you've made any changes you desire, type **b** to boot the entry.



When you edit a GRUB entry during the boot process, the changes you make are temporary. If you want to make permanent changes, you must edit the configuration file as described earlier in "Configuring GRUB."

Ordinarily, Linux boots into a full multiuser mode in which any authorized user may log in. Sometimes, though, you may want Linux to boot into a simpler mode in which just the `root` user may access the system. This single-user mode is useful for performing recovery and system maintenance tasks because the processes run by ordinary users, including system software that may run automatically when the system boots in the normal multiuser mode, can interfere with maintenance tasks such as low-level disk checks. To enter single-user mode, follow these steps:

1. Select the Linux entry you want to boot in the GRUB menu.
2. Type **e**, as described earlier, to edit the entry.
3. Select the line that begins with the word `kernel` and type **e** to edit it.
4. Use the arrow keys to move to the end of the line.
5. Press the spacebar, type **single**, and then press the Enter key to accept this change. (Typing **1** will also work.)
6. Type **b** to begin booting the system.

When the computer boots, it will boot directly into a text-mode root Bash prompt. You can use this prompt to change configuration files or otherwise manage the system. When you're done, you can reboot the system or type **telinit 2**, **telinit 3**, or **telinit 5** to start normal system services. (Which number you use depends on your distribution. The **telinit** command is described in more detail in Chapter 4, "Managing System Services.")



If your system boots normally, you may enter single-user mode from a normal boot by typing **telinit 1** at a root Bash prompt. Using GRUB to enter single-user mode is most useful if the system doesn't boot normally.

Using Rescue Discs

If Linux won't boot at all, even into single-user mode, you can use a Linux rescue disc to boot a working Linux installation and use it to recover your normal system. Rescue discs are available on CD-ROMs, DVD-ROMs, USB flash drives, and other removable media. They enable you to boot from the removable medium and make changes to your nonworking installation on your hard disk. Examples of Linux rescue discs include the following:

Your distribution's installation media Most distributions' installation media include some form of rescue mode. Check your documentation, or study the options presented by the boot disc when you boot it. Sometimes distributions provide a separate rescue disc, so you may want to check your distribution's Web site.

Knoppix This package, based at <http://www.knoppix.net>, is a full-fledged Linux installation based on Debian. You can download images that can be burned to CD-R or DVD-R media. Once booted, Knoppix is an unusually complete Linux system, although it's rather sluggish when run from optical media.

SystemRescueCd This package is a dedicated rescue system available as a CD-R image. In many respects it's similar to Knoppix, but it's targeted explicitly as a rescue system, whereas Knoppix serves a broader purpose. SystemRescueCd is based on Gentoo Linux. Check http://www.sysresccd.org/Main_Page for more details.



You can use a rescue disc that's based on a distribution other than the one you use—for instance, Knoppix works fine to recover a Fedora system. Some files' ownership may seem to change when you use a rescue disc, though. Don't try to correct such changes, since they're probably artifacts of different configurations of the rescue disc and your normal installation.

Once your rescue disc boots, you'll probably be presented with a root shell, in either a text-mode session or a GUI login. The rescue disc might or might not detect and mount your normal Linux system, so you may need to perform this task manually. (Chapter 6 describes how to do this.) Once your normal system is mounted, you can edit configuration files to overcome whatever problem is preventing your normal installation from booting.

Resetting the *root* Password

One common problem that may prompt use of single-user mode or a rescue disc is if you've forgotten the root password. If you can boot your system into single-user mode, you can reset the root password using the normal `passwd` utility:

```
# passwd
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
```

Chapter 5 describes the `passwd` utility in more detail. For now, know that you can use it to reset the root password; however, you must have root access to do so. Ordinarily, this requires you to have the root password. If you can enter single-user mode, though, you don't need the root password to reset it.

Another approach to resetting the root password is to edit the `/etc/shadow` file, which is described in more detail in Chapter 5. Locate the line that begins with `root` (normally the first line):

```
root:$1$6iwFIKHV$nDk0sd1bW2iGKsaAoxu87t:14329:0:99999:7:::
```

Delete all the text between the first and second colons (:) on this line. This string, which looks like gibberish, is the encrypted password. Deleting this string sets a null password so that no password is required to log into the account. Alternatively, you can copy the encrypted password from an account whose password you do remember to the root account's entry.



After setting a null password, particularly on the root account, you should *immediately* log in and set a real password on that account. Leaving a null password in place is a very serious security hole. Resetting the root password by directly editing `/etc/shadow` is handy if you need to boot into a rescue disc for some reason, but it's generally safer to boot into single-user mode and use the `passwd` command.

Examining Boot Messages with *dmesg*

Linux records data on various critical actions as it works. Some of these records end up in log files stored in the `/var/log` directory tree, as described in Chapter 4. One class of data is recorded in a different way, though, and this information is particularly relevant when you want to investigate boot problems. This information is generated by the kernel and is stored in the *kernel ring buffer*. You can examine this buffer with the `dmesg` command.

At a command prompt, type **dmesg** to see the contents of the ring buffer. Ordinarily, the result will be hundreds of lines of text. If you need to study the ring buffer in detail, you may want to employ the **less** pager:

```
$ dmesg | less
```

This command passes the output of **dmesg** through **less**, which is a program that enables you to page forward and backward through a long text file. (Both **less** and the pipe, **|**, are described in more detail in Chapter 2.)

The kernel ring buffer's contents change over time, so if you want to use **dmesg** to debug boot problems, you should use **dmesg** as soon after booting as possible. What you see will most likely seem cryptic at first. I recommend you examine the kernel ring buffer on a working Linux system to get a feel for what it contains. As you learn about Linux, its hardware, and how Linux names its hardware, the **dmesg** output will become less cryptic.

If you know or suspect that a problem is related to a particular hardware device or software configuration, you can search the ring buffer for information on that hardware or software. For instance, if a USB device isn't being detected, you could search for the string **USB** (or **usb**) or search for the name of the device or its driver. You may discover an error message that will point you toward a solution. Even the absence of information may be relevant; for instance, if you find no mention of USB devices, it could mean that the USB drivers aren't being loaded at all.

Configuring X

With Linux installed and booting, you can begin turning your attention to specific configuration tasks. One of the first of these tasks is setting up X. Although X usually works acceptably on a stock installation, sometimes you must change the X software you're using or tweak the X configuration. You should also know how to start and stop X, both temporarily and by setting a default mode.

In addition to the basics of X configuration, you should be familiar with the X environment. Typically, a *desktop environment* runs in X, providing you with an easy way to launch GUI programs, a file manager, and other tools. You should also be familiar with a tool that was mentioned earlier: an **xterm** or similar program that you can use to run text-mode programs in X.

Selecting an X Server

X is a network-enabled GUI system. It consists of an *X server*, which displays information on its local monitor and sends back user input from a keyboard and mouse; and an *X client*, which is a program that relies on the X server for user interaction. Although these two programs frequently run on the same computer, they don't need to do so. Chapter 9, "Configuring Advanced Networking," includes additional information on using X over a network. The

rest of this chapter assumes you'll be running X programs on the same system that runs the X server, but you don't install X differently if you'll be running X programs remotely.

The X server includes the driver for your video card, as well as support for your mouse and keyboard. Therefore, it's important that you know something about your video card when you install and configure your X server.

Determining Your Video Card Chipset

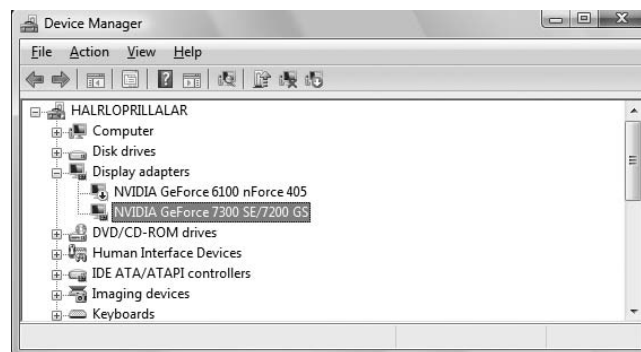
To properly configure X for your system, you must know what video chipset your system uses. Today, three companies—ATI, nVidia, and Intel—dominate the video chipset market, and most video cards and computers are marketed with fairly prominent claims of who made the video chipset. If you're using an oddball or older card, though, you may have trouble finding this information. You have several ways of approaching this problem:

Autodetection Linux can often autodetect the chipset, either during system installation or by running an X configuration tool after installation.

Video card documentation It's worthwhile to check the product's documentation. This documentation might not use the word “chipset,” though; it could use a phrase such as “powered by” or “based on.”

Windows driver report If the computer dual-boots to Windows or if you've just bought a Windows system and intend to convert it to Linux, you can use the Windows Control Panel to find out what video hardware is installed. In Windows Vista, double-click the Device Manager icon in the Control Panel. Click the plus sign next to the Display Adapters item. This will produce a list of the video cards installed in the computer, as shown in Figure 1.1. (Normally, there'll be just one, but Figure 1.1 shows a computer with two video cards: an nVidia GeForce 6100 and an nVidia GeForce 7300.) Double-click the entry for more information; this produces the Properties dialog box for the video card. The driver and manufacturer name may be that of the video card or of the chipset.

FIGURE 1.1 The Windows Device Manager may provide information on the video card hardware.



Linux identification tools The `dmesg` utility, described earlier, may provide clues as to the video card chipset. You can also type `lspci` at a command prompt to obtain identifying information on most of your installed hardware devices, including the video card. Both methods require you to wade through information on nonvideo devices.

One point to keep in mind when identifying the video card chipset is that video cards and video card chipsets are often made by different manufacturers. For instance, nVidia produces chipsets that are used in boards made by ASUS, Biostar, Gigabyte, and others. Linux and X don't care about who made the video card; only the chipset manufacturer is important.



NOTE

Increasingly, video functions are built into computer motherboards. Nonetheless, it's still common to refer to "video cards" as if they were physically distinct cards, as they once universally were.

Choosing an X Server

All major Linux distributions ship with a free X server. In the past, a server known as XFree86 was common, but most distributions have switched to X.org-X11 instead, because of changes to the XFree86 licensing terms. These two servers are very similar, though; X.org-X11 6.7.0 was based on XFree86 4.3.99. You can learn more about XFree86 at <http://www.xfree86.org>, and X.org-X11 is headquartered at <http://www.x.org>. As I write, the current versions are XFree86 4.8.0 and X.org-X11 7.4.

Linux distributions from 2001 and before used XFree86 3.3.6 or earlier, but more recent distributions use XFree86 4.x or X.org-X11. Some major architectural modifications marked the change to XFree86 4.x, and some configuration files changed with this release. By the time X.org-X11 was forked off the XFree86 project, XFree86 3.3 had become largely obsolete. Thus, I don't cover this old version of XFree86. If you encounter it or must use it because of poor support for an obscure video card in more recent X servers, though, you should be aware that some configuration options changed between XFree86 3.3.6 and 4.0.

Some video card and chipset manufacturers have made XFree86- and X.org-X11-compatible drivers available for their products. Thus, it's worth checking the Web sites maintained by your board and chipset manufacturers to see if drivers are available. This is definitely true if the main XFree86 or X.org-X11 release doesn't include appropriate drivers, and it may be true even if there are drivers—the manufacturers' offerings often offer improved performance, particularly in the realms of 3D and full-motion video acceleration.

XFree86 or X.org-X11 occasionally doesn't support a device at all. You have three choices in this case:

Use the frame buffer device. The Linux kernel has some video drivers of its own. These can be accessed via the *frame buffer* X driver. For this to work, your kernel must include frame buffer support for your video chipset.

Use another X server. It's conceivable that XFree86 will work where X.org-X11 doesn't, or vice versa. In addition, a company called Xi Graphics (<http://www.xig.com>) produces a commercial X server for Linux, known as Accelerated-X. This server occasionally works on hardware that's not supported by XFree86 or X.org-X11, and sometimes it produces better speed.

Replace the hardware. If you have a recalcitrant video card, the final option is to replace it. You may be able to swap with a Windows system that uses a different card, or you may need to buy a new card. Unfortunately, this isn't always an option; you can't replace the video card on a notebook computer, for instance.

Installing an X Server or Driver

Actually installing an X server is usually not very difficult; it's a matter of using your distribution's package management tools to install the software, much as you would any other software (described in Chapter 7). In most cases, this will be done during system installation. You'll have to manually install a server only if you failed to install X during system installation or if you need to install a new server.



X normally comes in several packages. Only one package contains the X server proper; others provide support libraries, fonts, utilities, and so on.

One server package supports all video chipsets. The name of this package varies from one distribution to another, but it's likely to be called XFree86, XFree86-server, xserver-xfree86, or something similar for XFree86; or xorg-x11 or something similar for X.org-X11. Consult Chapter 7 for details of how to locate and install packages.

The main X server program is called X or Xorg, which is usually stored in `/usr/X11R6/bin` or `/usr/bin`. This program is a generic X server. It relies on separate driver modules, which are installed along with the main package in most cases.

If you're using an X driver provided by a video card manufacturer, follow the manufacturer's directions for installing the driver. In most cases you'll be told to run a program that you download from the manufacturer's Web site. Some distributions provide packages with these drivers so you can install them more easily.

Setting Up X

XFree86 is configured through the XF86Config file, which is usually located in `/etc` or `/etc/X11`. For XFree86 4.x, this file is sometimes called XF86Config-4. X.org-X11 calls its configuration file `xorg.conf`; it's located in the same location and has the same format. (For simplicity, I refer to both files as `xorg.conf` from now on.) Consult your server's documentation if you're using something other than X.org-X11 or XFree86.

Configuring X requires editing the configuration file in any of several ways. You can adjust settings related to input devices (the keyboard and mouse), the video card, and the monitor.

Methods of Configuring X

XFree86 can be configured via either of two methods: by using configuration tools and by configuring manually. Configuration tools prompt you for information or obtain it directly from the hardware and then write the `xorg.conf` file, which is a standard plain-text file like other Linux configuration files. Because this file is relatively complex, it's usually wise to begin with an automatic configuration, even if it's a flawed one. Manual configuration involves opening `xorg.conf` in a text editor and changing its settings using your own know-how. You can use this method to tweak a working configuration for better performance or to correct one that's not working at all. Either way, you may need to configure X, test it, reconfigure X, test it, and so on, for several iterations until you find a configuration that works correctly. (The upcoming section "Starting and Stopping X" describes how to start or restart X to test a new configuration.)

Several utilities can help in X configuration:

The X server The XFree86 or Xorg server itself includes the capacity to query the hardware and produce a configuration file. To do so, type **XFree86 -configure** or **Xorg -configure** when no X server is running. The result should be a file called `/root/XF86Config.new` or `/root/xorg.conf.new`. This file might not produce optimal results, but it is at least a starting point for manual modifications.

Distribution-specific tools Many modern distributions ship with their own custom X configuration tools. These tools frequently resemble the distribution's install-time X configuration tools, which can vary substantially. Increasingly, these tools rely on automatic X detection of hardware and settings, so these tools can be extremely limited; they may give only screen resolution and color depth options, for instance. They can usually be accessed from the desktop's menu system.

In the past, tools such as `xf86config`, `Xconfigurator`, and `XF86Setup` could be used to configure X. These programs are seldom installed on modern systems, though. Therefore, if you need to perform serious changes to your X configuration, chances are you'll need to edit `xorg.conf` in a text editor.

The `xorg.conf` file consists of a number of labeled sections, each of which begins with the keyword `Section`, followed by the section name in quotes, and ends with the keyword `EndSection`. Between these two lines are lines that define features relevant to the configuration of that feature. There may also be comments, which are lines that begin with hash marks (`#`). For instance, here's a section that defines where the computer can find certain critical files:

```
Section "Files"
    RgbPath    "/usr/X11R6/lib/X11/rgb"
    # Multiple FontPath entries are allowed
    FontPath   "/usr/X11R6/lib/X11/fonts/75dpi"
    FontPath   "/usr/X11R6/lib/X11/fonts/Type1"
EndSection
```



If you have a working configuration, be sure to back up `xorg.conf` before modifying it. If you mistakenly delete or modify some critical line, you can easily end up with a system that won't start X, and without a backup, it can be difficult to restore even a partially functioning system.

Many `xorg.conf` sections include `Identifier`, `ModelName`, `VendorName`, or `BoardName` lines. The `Identifier` provides a name for the section that can be used by other sections to refer to the first one. The `ModelName`, `VendorName`, or `BoardName` line, if present, is intended for human consumption, so you can put anything there you like. A `Driver` line, by contrast, points X to a driver for the device. This is extremely important, so you shouldn't change it unless you're positive that the current entry is wrong.

Setting Miscellaneous Options

Some sections of the `xorg.conf` file relate to miscellaneous options or those that require just a handful of lines. Nonetheless, getting these settings right is important to a functioning X system. Specific sections in this category include the following:

Files The `Files` section hosts information on the locations of important files. The entries you're most likely to change relate to the locations of X's fonts. These are handled through the `FontPath` option line. Modern distributions often omit this section and instead rely on default settings. If you need to add fonts to your system, though, you may need to add `FontPath` entries that point to your existing and new font directories.

The keyboard One `InputDevice` section defines the operation of the keyboard in X. This section normally has a `Driver "kbd"` line. In most cases, the default settings (or those set automatically based on your install-time choices) work fine. You may want to adjust the `XkbLayout` or `XkbModel` option to use a different layout or model if these features were set incorrectly. The `AutoRepeat` option sets the delay before keyboard repeat begins and the repeat rate, both in milliseconds (thousandths of a second). This feature is usually overridden in desktop environment configurations, so it may not have any practical effect.

The mouse A second `InputDevice` section, with a `Driver "mouse"` line, defines the mouse. The default settings autodetect the mouse, which works for the vast majority of PS/2 and USB mice. If you use a particularly obscure model, you may need to set `Protocol` and `Device` options, as in `Option "Protocol" "Logitech"` or `Option "Device" "/dev/ttyS1"`.



X programs frequently use the middle button; for instance, text editors use it for pasting text. Therefore, any Linux workstation should be equipped with a genuine three-button mouse rather than a two-button device. Scroll wheels on mice that are so equipped can usually function as a middle button, as well as handling wheel duty. The `Option "Emulate3Buttons" "yes"` option enables you to use a two-button mouse in Linux, but doing so is awkward.

Setting Monitor Options

Some of the trickiest aspects of X configuration relate to the monitor options. You set these in the Monitor section, which looks like this:

```
Section "Monitor"
    Identifier "Iiyama"
    ModelName "VisionMaster Pro 450"
    HorizSync 27.0-115.0
    VertRefresh 50.0-160.0
    # My custom 1360x1024 mode
    Modeline "1360x1024" 197.8 \
        1360 1370 1480 1752 \
        1024 1031 1046 1072 -HSync -VSync
EndSection
```

The HorizSync and VertRefresh lines are extremely critical; they define the range of horizontal and vertical refresh rates that the monitor can accept, in kilohertz (kHz) and hertz (Hz), respectively. Together, these values determine the maximum resolution and refresh rate of the monitor. X won't exceed these limits, since doing so can theoretically damage the monitor. (All monitors made since the mid-1990s have circuitry to protect them from such abuse, so this concern isn't as important as it once was.)

Some X configuration utilities show a list of monitor models or resolution and refresh rate combinations (such as "1024 × 768 at 72 Hz") to obtain this information. This approach is often simpler to handle, but it's less precise than entering the exact horizontal and vertical sync values.

To settle on a resolution, X looks through a series of *mode lines*, which are specified via the Modeline option. Computing mode lines is tricky, so I don't recommend you try it unless you're skilled in such matters. The mode lines define combinations of horizontal and vertical timing that can produce a given resolution and refresh rate. For instance, a particular mode line might define a 1024 × 768 display at a 90Hz refresh rate, and another might represent 1024 × 768 at 72Hz.

When asked to produce a given resolution, X searches all the mode lines that accomplish the job, discards those that the monitor can't handle, and uses the remaining mode line that creates the highest refresh rate at that resolution. (If no mode line supports the requested resolution, X drops down to another specified resolution and tries again.)

Modeline entries were common in XFree86 3.3.x. Although they're still supported in XFree86 4.x and X.org-X11, these versions of X include standard mode lines that obviate the need for Modeline entries in the configuration file unless you want to use an unusual resolution or refresh rate.

Setting Video Card Options

XFree86 4.x and X.org-X11 use driver modules that are stored in separate files from the main X server executable. You must tell the server which driver module to use in

the `xorg.conf` file. In particular, the driver module is set by a line in the Device section, which resembles the following:

```
Section "Device"
    Identifier "On-Board Video"
    VendorName "ATI"
    BoardName "Radeon HD3200"
    Driver "fglrx"
    BusID "PCI:1:5:0"
EndSection
```

The Driver line is the most important one in this section. Driver files reside in an X drivers directory, such as `/usr/X11R6/lib/modules/drivers/` or `/usr/lib/xorg/modules/drivers/`. Most of the drivers' filenames end in `_drv.o`, and if you remove this portion, you're left with the driver name. For instance, `fglrx_drv.o` corresponds to the `fglrx` driver.

The BusID line in this example uniquely identifies the video card by the slot in which it's inserted. (This number is fixed in the case of video hardware built into the motherboard.) Using this line may be necessary if your computer has two video cards, particularly if they're from the same manufacturer.

Many drivers support additional driver-specific options. Consult the `xorg.conf` man page or other driver-specific documentation for details.

Setting Screen Options

The Screen section ties together the other sections. Here's a short example:

```
Section "Screen"
    Identifier "screen1"
    Device "On-Board Video"
    Monitor "Iiyama"
    DefaultDepth 16
    Subsection "Display"
        Depth 8
        Modes "1280x1024" "1024x768" "640x400"
    EndSubsection
    Subsection "Display"
        Depth 16
        Modes "1024x768" "800x600" "640x480"
        Virtual 1280 1024
        ViewPort 0 0
    EndSubsection
EndSection
```

Several key points in this section should be emphasized:

- The `Identifier` specifies an overall configuration. A configuration file can hold multiple `Screen` sections, as described shortly.
- The `Device` and `Monitor` lines point to specific `Device` and `Monitor` sections, respectively.
- The `DefaultDepth` line specifies the number of bits per pixel to be used by default. For instance, the preceding example sets this value to 16, so a 16-bit color depth is used, resulting in 2^{16} , or 65,536, possible colors.
- Each `Subsection` defines a particular display type. They have associated color depths (specified by the `Depth` line) and a series of resolutions (specified by the `Modes` line). The system tries each resolution specified by the `Modes` line in turn, until it finds one that works. There are also various optional parameters, such as `Virtual` (which defines a virtual screen that can be larger than the one that's actually displayed) and `Viewport` (a point within that virtual display at which the initial display is started).

One final section is required: the `ServerLayout` section. This section consists of lines that identify the default `Screen` section and link it to mouse and keyboard definitions. For instance, a typical configuration will include a `ServerLayout` section resembling the following:

```
Section "ServerLayout"
    Identifier   "layout1"
    Screen      "screen1"
    InputDevice "Mouse1"  "CorePointer"
    InputDevice "Keyboard1" "CoreKeyboard"
EndSection
```



Although I describe the `ServerLayout` section last because it ties together all the other sections, it can appear earlier in the file—perhaps even first. The order of sections in the `xorg.conf` file is arbitrary.

Normally, an `xorg.conf` file will have just one `ServerLayout` section, but by passing the `-layout name` parameter to the server program, you can tell the server to use a different `ServerLayout` section, if one is present. You might use this to start X using a different mouse, for instance—say, a USB mouse on a notebook rather than the built-in PS/2 touch pad.

Managing GUI Logins

Linux can boot into a purely text-based mode in which the console supports text-based logins and text-mode commands. This configuration is suitable for a system that runs as a server computer or for a desktop system for a user who dislikes GUIs. Most desktop users, though, expect their computers to boot into a friendly GUI. For such users, Linux supports a login system that starts X automatically and provides a GUI login screen. Configuring and managing this system requires you to understand a bit of how it works, how to run it, and how to change the configuration.

Understanding GUI Logins

X is a network-enabled GUI. This fact has many important consequences, and one of these relates to Linux's GUI login system. This system employs a network login protocol, the *X Display Manager Control Protocol (XDMCP)*. To handle remote logins, an XDMCP server runs on a computer and listens for connections from remote computers' X servers. To handle local logins, an XDMCP server runs on a computer and starts the local computer's X server. The XDMCP server then manages the local X server's display—that is, it puts up a login prompt like that shown in Figure 1.2.

FIGURE 1.2 An XDMCP server manages local GUI logins to a Linux system.



Three XDMCP servers are common on Linux: the X Display Manager (XDM), the KDE Display Manager (KDM), and the GNOME Display Manager (GDM). A few more XDMCP servers are also available, but these three are the most important. As you may guess by their names, KDM and GDM are associated with the KDE and GNOME projects, respectively, but neither limits your choice of desktop environment. Most Linux distributions run either GDM or KDM as the default XDMCP server, but you can change which one your system uses if you don't like the default.

Running an XDMCP Server

Several methods exist to start an XDMCP server. These two are the most common:

- Launching the XDMCP server more or less directly from `init`, via an entry in `/etc/inittab` or its ancillary configuration files. (The `init` program is the first one the kernel launches; it controls the rest of the system startup process through various means.)
- Launching the XDMCP server as part of a runlevel's startup script set, via a SysV startup script. (A runlevel is a set of programs that run concurrently. Several runlevels exist so as to support multiple configurations—such as booting with or without an XDMCP server.)

Chapter 4 describes both `init` and SysV startup scripts in general, so consult it for information about these processes.

Whichever method is used, many distributions configure themselves to run their chosen XDMCP server when they start in runlevel 5 but not when they start in runlevel 3. This is the only difference between these two runlevels in most cases. Thus, changing from runlevel 3 to runlevel 5 starts X and the XDMCP server on many distributions, and switching back to runlevel 3 stops X and the XDMCP server. As described in more detail in Chapter 4, you can change runlevels as root with the `telinit` command:

```
# telinit 5
```

Permanently changing the runlevel requires editing the `/etc/inittab` file and, in particular, its `id` line:

```
id:5:initdefault:
```

Change the number (5 in this case) to the runlevel you want to use as the default.

A few distributions—most notably Gentoo, Debian, and Debian's derivatives (including the popular Ubuntu)—attempt to start an XDMCP server in all runlevels (or don't do so at all). This is done through the use of a SysV startup script called `xdm`, `kdm`, or `gdm`. Thus, you can temporarily start or stop the XDMCP server by running this script and passing it the `start` or `stop` option. To permanently enable or disable the XDMCP server, you should adjust your SysV startup scripts, as described in Chapter 5.

In addition to the question of whether to run an XDMCP server is the question of *which* XDMCP server to run. Most distributions set a default XDMCP server in one way or another. Two common methods exist:

Selection via configuration file Some distributions hide the XDMCP server choice in a configuration file, often in the `/etc/sysconfig` directory. In Fedora, the `/etc/sysconfig/desktop` file, if present, sets the `DISPLAYMANAGER` variable to `XDM`, `KDM`, or `GDM`. In OpenSUSE, `/etc/sysconfig/displaymanager` sets the `DISPLAYMANAGER` variable in a similar way, but using lowercase display manager names.

Selection via SysV script In Debian and derivative distributions, such as Ubuntu, the display manager is set via choice of SysV startup script—use the `gdm` script to use GDM, `kdm`

to use KDM, and so on. By default, only one XDMCP server (and associated SysV startup script) is installed, so if you want to change your XDMCP server, you may need to install your desired server. Chapter 4 describes how to configure specific SysV startup scripts to run automatically.

Unfortunately, distribution maintainers have had a habit of changing the details of how XDMCP servers are launched from time to time, and the settings are often buried in poorly documented configuration files. Thus, you may need to go digging through the files in your `/etc` directory to find the correct setting.

Configuring an XDMCP Server

XDMCP servers, like most programs, can be configured. Unfortunately, this configuration varies from one server to another, although there are some commonalities.

Configuring XDM

XDM is the simplest of the major XDMCP servers. It accepts usernames and passwords but doesn't enable users to perform other actions, such as choose which desktop environment to run. (This must be configured through user login files.)

XDM's main configuration file is `/etc/X11/xdm/xdm-config`. Most distributions ship with a basic `xdm-config` file that should work fine for a local workstation. You can edit this and other XDM configuration files; however, chances are you won't have to do so. One case when you would want to make changes is to enable remote network access to X. This topic is covered in Chapter 10, "Configuring Network Servers."

Configuring KDM

KDM is based partly on XDM and so shares many of its configuration options. Unfortunately, the location of the KDM configuration files is unpredictable; sometimes KDM uses the XDM configuration files, other times they're stored in `/etc/X11/kdm` or `/etc/kde/kdm`, and sometimes they're stored in a truly strange location such as `/usr/share/kde4/config/kdm/kdmrc`.



If you can't find the KDM configuration files, try using your package management tools, described in Chapter 7. Try obtaining lists of files in the `kdebase` package or other likely candidates, and look for the KDM configuration files.

KDM expands on XDM by enabling users to select a session type when they log in, to shut down the computer from the main KDM prompt, and so on. Most of these extra options are set in the `kdmrc` file, which appears in the same directory as the other KDM configuration files. Some of these options override the more common XDM configuration options for the same features.

Configuring GDM

GDM is more of a break from XDM than is KDM. GDM doesn't use the conventional XDM configuration files or similar files. Instead, it uses configuration files that are usually stored in `/etc/X11/gdm` or `/etc/gdm`. The most important of these files is `gdm.conf`, and it has a format

similar to the `kdmrc` file. (Recent versions of GDM use a file called `custom.conf`, which holds only the overrides of default settings stored elsewhere.)

Like KDM, GDM provides extra options over those of XDM. These options include the ability to choose your login environment and shut down the computer. GDM is a bit unusual in that it prompts for the username and only then presents a prompt for the password. (The GDM username prompt was shown in Figure 1.2 earlier in the chapter.) XDM and KDM both present fields for the username and password simultaneously.

Using Window Managers and Desktop Environments

X is a fairly bare-bones environment; it can display windows without borders, it can display text or graphics in those windows, and it can handle some fairly minimal operations beyond that. X does not itself provide menus, buttons, file managers, or other advanced tools. These features are provided by other tools, known as *window managers* and *desktop environments*.

A window manager provides decorative and functional borders around X windows. When you resize or drag a window, it's the window manager that you're using. Most window managers also control the *root window*—that is, the screen as a whole. If you right-click the desktop's background, chances are you'll see a menu pop up; that's a window manager tool. Common window managers include `fvwm`, `twm`, `IceWM`, `Blackbox`, and `Metacity`.

Some minimalist Linux users run a window manager in their X sessions but little else, aside from whatever programs they actively use. Most users, though, run a desktop environment atop the window manager. This is a set of software tools that facilitates launching programs, adjusting user interface settings, and so on. Desktop environments also typically include a file manager, which provides drag-and-drop file manipulation. The most popular Linux desktop environments are the GNU Network Object Model Environment (GNOME), the K Desktop Environment (KDE), and `Xfce`.

If you use KDM or GDM as your XDMCP server, you can select which desktop environment to use when you log in. Look for a menu or option button that enables you to choose the session type. On modern distributions, installing a window manager or desktop environment automatically adjusts the XDM and GDM configurations to present the new environment as a login option.

Once you've logged in, the desktop environment will present menu bars and options that are similar to those of Windows or Mac OS. You'll be able to launch programs by picking them out of menus, you'll be able to access your disks, and so on.

Using Terminal Programs

One of the most important programs you can access from an X session is an `xterm` program or something similar. These programs are sometimes referred to collectively as *terminal programs* or *consoles*. They enable you to run text-mode commands and programs from within a GUI environment. Since so many Linux tools are text-based, knowing how to use a terminal program is a critical skill for any Linux administrator.

**NOTE**

The term *terminal program* is often applied to a second type of program. This type of terminal program opens a text-mode data transfer session over a communications device, such as an RS-232 serial port or a modem. In years past, terminal programs of this type were commonly used for communicating with remote computers or for linking two or more nearby computers together. With the advent of modern networks and the Internet, terminal programs of this type have become much less important, so the words *terminal program* now usually refer to xterm-type programs.

The main trick in running a terminal program is to find one. Most Linux window managers and desktop environments provide one or more entries in their program lists to launch terminal programs. Check your menus, and particularly any submenus entitled Accessories, System, or System Tools, for entries called Terminal, Konsole, or xterm. These are the names of three common terminal programs.

If you can't find an entry for a terminal program, try to find an entry entitled Run, Run Other, or something similar. Such an entry should enable you to run an arbitrary program. Type **terminal**, **konsole**, or **xterm** into its text-entry field to launch a terminal.

Once it's running, a terminal program normally launches a *shell*, which is a Linux tool for entering text-mode commands. (Chapter 2 describes shells in more detail.)

Managing Hardware

Most hardware manufacturers ensure that their products work properly under Windows. Some do the same for Linux, but you're usually on your own when you use Linux. Fortunately, tools and resources exist to help with these tasks. You can check for hardware compatibility before you even buy it, using Internet resources. Once hardware is installed, Linux provides tools to help you identify that hardware and to configure it properly.

Finding Compatible Hardware

When you shop for hardware for use on a Linux computer, you must remember that not all hardware works with Linux. Several resources exist to help in this regard:

The Hardware Compatibility List (HCL) This resource, located at <http://www.linuxquestions.org/hcl>, is the closest thing to a comprehensive source for Linux hardware compatibility. You can use it to check on the compatibility of a specific product or to find products in a category that are known to be compatible.

The OpenPrinting Database This site (http://www.linuxprinting.org/prINTER_list.cgi) maintains a list of printers along with comments on their compatibility with Linux.

The SANE Database The Scanner Access Now Easy (SANE) project maintains a database of scanners, with information on their compatibility with the SANE software, which is the main scanner package for Linux. Check <http://www.sane-project.org> for more details.

The ALSA Project The Advanced Linux Sound Architecture (ALSA) project maintains the mainstream Linux sound drivers. Its Web page, <http://alsa-project.org>, includes notes on compatibility with specific devices.

These databases are maintained by driver developers and end users. As such, they're necessarily incomplete and therefore of limited value. Even limited value is better than no value, though, so you should definitely check these sites before buying hardware or to evaluate the compatibility of hardware you already own.

Another resource you may want to consult is your hardware's manufacturer. Check both the Web site for the company that made the product and, if you know it, the company that made the chipset that drives the product. To Linux, the chipset is what's important, but the hardware's manufacturer may have information or even drivers available for download.

Broadly speaking, products that are most likely to give problems are those that include circuitry that requires specialized drivers—video cards, network cards, scanners, printers, and so on. Some products, by contrast, are extremely standardized and so seldom cause problems. Examples in this category include RAM, hard disks, CD-ROM and DVD-ROM drives, keyboards, mice, and monitors.

Identifying Hardware in Linux

If you've installed Linux on a computer and aren't sure what hardware is available or if you've just installed a device and want to verify that it's accessible, you can use various Linux tools to help identify the hardware. Tools to identify PCI cards and USB devices exist. You can also query the kernel drivers that are loaded and examine a special directory to locate hardware information.

Identifying PCI Devices

The Peripheral Component Interconnect (PCI) standard defines a physical and logical set of parameters that enable plug-in cards to be used in any PCI-supporting computer. PCI devices plug into the computer's motherboard or are built into the motherboard directly. In Linux, the `lspci` command displays information on PCI devices. Table 1.1 summarizes the options to this command.

TABLE 1.1 Options for `lspci`

Option	Effect
-v	Increases verbosity of output. This option may be doubled (-vv) or tripled (-vvv) to produce yet more output.
-n	Displays information in numeric codes rather than translating the codes to manufacturer and device names.

TABLE 1.1 Options for `lspci` (*continued*)

Option	Effect
<code>-nn</code>	Displays both the manufacturer and device names and their associated numeric codes.
<code>-x</code>	Displays the PCI configuration space for each device as a hexadecimal dump. This is an extremely advanced option. Tripling (<code>-xxx</code>) or quadrupling (<code>-xxxx</code>) this option displays information about more devices.
<code>-b</code>	Shows IRQ numbers and other data as seen by devices rather than as seen by the kernel.
<code>-t</code>	Displays a tree view depicting the relationship between devices.
<code>-s [[[[<i>domain</i>]:]<i>bus</i>]:] [<i>slot</i>][.<i>func</i>]]</code>	Displays only devices that match the listed specification.
<code>-d [<i>vendor</i>]:[<i>device</i>]</code>	Shows data on the specified device.
<code>-i <i>file</i></code>	Uses the specified file to map vendor and device IDs to names. (The default is <code>/usr/share/misc/pci.ids</code> .)
<code>-m</code>	Dumps data in a machine-readable form, intended for use by scripts. A single <code>-m</code> uses a backward-compatible format, whereas doubling (<code>-mm</code>) uses a newer format.
<code>-D</code>	Displays PCI domain numbers. These numbers normally aren't displayed.
<code>-M</code>	Performs a scan in bus-mapping mode, which can reveal devices hidden behind a misconfigured PCI bridge. This is an advanced option that can be used only by root.
<code>--version</code>	Displays version information.

Identifying USB Devices

Universal Serial Bus (USB) devices normally attach externally to the computer. You can check to see what USB devices are connected using the `lsusb` command, which is similar in many ways to `lspci`. Table 1.2 summarizes `lsusb` options.

TABLE 1.2 Options for `lsusb`

Option	Effect
<code>-v</code>	Increases verbosity of output
<code>-t</code>	Displays a tree view depicting the relationship between devices
<code>-s [[bus]:][devnum]</code>	Displays only devices that match the listed specification
<code>-d [vendor]:[device]</code>	Shows data on the specified device
<code>-D device</code>	Displays information on the specified <i>device</i> , which is a device file in the <code>/dev</code> directory tree
<code>--version</code> or <code>-V</code>	Displays version information

Note that `lsusb` displays information on both the devices that are attached to your computer and on the USB controller in the computer itself.

Identifying Kernel Drivers

Hardware in Linux is handled by kernel drivers, many of which come in the form of *kernel modules*. These are stand-alone driver files, typically stored in the `/lib/modules` directory tree, that can be loaded and unloaded to provide access to hardware. Typically, Linux loads the modules it needs when it boots, but you may need to load additional modules yourself.

You can learn about the modules that are currently loaded on your system by using `lsmod`, which takes no options and produces output like this:

```
$ lsmod
Module                Size  Used by
isofs                 35820  0
zlib_inflate          21888  1 isofs
floppy                65200  0
nls_iso8859_1          5568  1
nls_cp437              7296  1
vfat                  15680  1
fat                   49536  1 vfat
sr_mod                19236  0
ide_cd                42848  0
cdrom                  39080  2 sr_mod,ide_cd
```



NOTE

The example output for `lsmod` has been edited for brevity. Although outputs this short are possible with certain configurations, they're rare.

The most important column in this output is the first one, labeled `Module`; this column specifies the names of all the modules that are currently loaded. You can learn more about these modules with `modinfo`, as described shortly, but sometimes their purpose is fairly obvious. For instance, the `floppy` module provides access to the floppy disk drive.

The `Used by` column of the `lsmod` output describes what's using the module. All the entries have a number, which indicates the number of other modules or processes that are using the module. For instance, in the preceding example, the `isofs` module (used to access CD-ROM filesystems) isn't currently in use, as revealed by its 0 value; but the `vfat` module (used to read VFAT hard disk partitions and floppies) is being used, as shown by its value of 1. If one of the modules is being used by another module, the using module's name appears in the `Used by` column. For instance, the `isofs` module relies on the `zlib_inflate` module, so the latter module's `Used by` column includes the `isofs` module name. This information can be useful when you're managing modules. For instance, if your system produced the preceding output, you couldn't directly remove the `zlib_inflate` module because it's being used by the `isofs` module; but you could remove the `isofs` module, and after doing so you could remove the `zlib_inflate` module. (Both modules would need to be added back to read most CD-ROMs, though.)



The `lsmod` command displays information only about kernel modules, not about drivers that are compiled directly into the Linux kernel. For this reason, a module may need to be loaded on one system but not on another to use the same hardware because the second system may compile the relevant driver directly into the kernel.

Using the `/proc` Filesystem

Linux uses a special filesystem, `/proc`, to control and provide information about much of the hardware on the computer. Although `lspci`, `lsusb`, `lsmod`, and some other tools provide useful information about specific subsystems, you can use `/proc` to obtain information on still more hardware. For instance, the `/proc/scsi` subdirectory hosts information on SCSI devices (as well as devices that look like SCSI devices, such as most SATA disks), `/proc/cpuinfo` delivers information on your CPU, and `/proc/interrupts` displays the interrupts used by hardware devices.

You may want to peruse your `/proc` filesystem to see what sorts of information it can provide. You can use the `cat` command to display the contents of a file, as in **`cat /proc/interrupts`**. Be aware that many of the files contained in this directory tree hold extremely technical information that may be mystifying unless you have a deep understanding of the hardware involved.



Don't try to modify the files in `/proc`. Writing to these files can cause your hardware to malfunction. In extreme cases you could wipe out your Linux installation!

Managing Kernel Modules

The `lsmod` command, described earlier, tells you what kernel modules are installed; however, you may need to load kernel modules, remove them, or configure how they operate. To perform these tasks, you must use other tools, such as `insmod`, `modprobe`, and `rmmod`.

Loading Kernel Modules

Linux enables you to load kernel modules with two programs: `insmod` and `modprobe`. The `insmod` program inserts a single module into the kernel. This process requires you to have already loaded any modules on which the module you're loading relies. The `modprobe` program, by contrast, automatically loads any depended-on modules and so is generally the preferred way to do the job.



In practice, you may not need to use `insmod` or `modprobe` to load modules because Linux can load them automatically. This ability relies on the kernel's module autoloader feature, which must be compiled into the kernel, and on various configuration files, which are also required for `modprobe` and some other tools. Using `insmod` and `modprobe` can be useful for testing new modules or for working around problems with the autoloader, though.

In practice, `insmod` is a fairly straightforward program to use; you type it followed by the module filename:

```
# insmod /lib/modules/2.6.29/kernel/drivers/block/floppy.ko
```

This command loads the `floppy.ko` module, which you must specify by filename. Modules have module names, too, which are usually the same as the filename but without the extension, as in `floppy` for the `floppy.ko` file.

You can pass additional module options to the module by adding them to the command line. Module options are highly module-specific, so you must consult the documentation for the module to learn what to pass. Examples include options to tell an RS-232 serial port driver what interrupt to use to access the hardware or to tell a video card framebuffer driver what screen resolution to use.

Some modules depend on other modules. In these cases, if you attempt to load a module that depends on others and those other modules aren't loaded, `insmod` will fail. When this happens, you must either track down and manually load the depended-on modules or use `modprobe`. In the simplest case, you can use `modprobe` much as you use `insmod`, by passing it a module name:

```
# modprobe floppy
```

As with `insmod`, you can add kernel options to the end of the command line. Unlike `insmod`, you specify a module by its module name rather than its module filename when you use `modprobe`. This helps make `modprobe` easier to use, as does the fact that `modprobe`

automatically loads dependencies. This greater convenience means that `modprobe` relies on configuration files. It also means that you can use options (placed between the command name and the module name) to modify `modprobe`'s behavior, as summarized in Table 1.3.

TABLE 1.3 Options for `modprobe`

Option	Effect
<code>-v</code> or <code>--verbose</code>	This option tells <code>modprobe</code> to display extra information about its operations. Typically, this includes a summary of every <code>insmod</code> operation it performs.
<code>-C filename</code>	The <code>modprobe</code> program uses a configuration file called <code>/etc/modprobe.conf</code> . You can change the file by passing a new file with this option, as in <code>modprobe -C /etc/mymodprobe.conf floppy</code> .
<code>-n</code> or <code>--dry-run</code>	This option causes <code>modprobe</code> to perform checks and all other operations <i>except</i> the actual module insertions. You might use this option in conjunction with <code>-v</code> to see what <code>modprobe</code> would do without loading the module. This may be helpful in debugging problems.
<code>-r</code> or <code>--remove</code>	This option reverses <code>modprobe</code> 's usual effect; it causes the program to remove the specified module and any on which it depends. (Depended-on modules are <i>not</i> removed if they're in use.)
<code>-f</code> or <code>--force</code>	This option tells <code>modprobe</code> to force the module loading even if the kernel version doesn't match what the module expects. This action is potentially dangerous, but it's occasionally required when using third-party binary-only modules.
<code>--show-depends</code>	You can see all the modules on which the specified module depends by using this option. It doesn't install any of the modules; it's purely informative in nature.
<code>-l</code> or <code>--list</code>	This option displays a list of available options whose names match the wildcard you specify. For instance, typing <code>modprobe -l v*</code> displays all modules whose names begin with <code>v</code> . If you provide no wildcard, <code>modprobe</code> displays all available modules. Like <code>--show-depends</code> , this option doesn't cause any modules to be loaded.



Table 1.3 is incomplete. Other `modprobe` options are relatively obscure, so you're not likely to need them often. Consult the `modprobe` man page for more information.

Kernel modules can take options that modify their behavior. These options can be specified in the `/etc/modules.conf` file. (Some distributions use files in `/etc/modules.d` instead.)

Removing Kernel Modules

In most cases, you can leave modules loaded indefinitely; the only harm that a module does when it's loaded but not used is to consume a small amount of memory. (The `lsmod` program shows how much memory each module consumes.) Sometimes, though, you may want to remove a loaded module. Reasons include reclaiming that tiny amount of memory, unloading an old module so you can load an updated replacement module, and removing a module that you suspect is unreliable.

The work of unloading a kernel module is done by the `rmmmod` command. This command takes a module name as an option:

`rmmmod floppy`

This example command unloads the `floppy` module. You can modify the behavior of `rmmmod` in various ways, as summarized by Table 1.4. A few more `rmmmod` options exist; consult the `rmmmod` man page for details.

TABLE 1.4 Options for `rmmmod`

Option	Effect
<code>-v</code> or <code>--verbose</code>	This option tells <code>rmmmod</code> to display extra information about its operations.
<code>-f</code> or <code>--force</code>	This option forces module removal even if the module is marked as being in use. Naturally, this is a very dangerous option, but it's sometimes helpful if a module is misbehaving in some way that's even more dangerous. This option has no effect unless the <code>CONFIG_MODULE_FORCE_UNLOAD</code> kernel option is enabled.
<code>-w</code> or <code>--wait</code>	This option causes <code>rmmmod</code> to wait for the module to become unused, rather than return an error message, if the module is in use. Once the module is no longer being used (say, after a floppy disk is unmounted if you try to remove the <code>floppy</code> module), <code>rmmmod</code> unloads the module and returns. Until then, <code>rmmmod</code> doesn't return, making it look like it's not doing anything.

Like `insmod`, `rmmmod` operates on a single module. If you try to unload a module that's depended on by other modules or is in use, `rmmmod` will return an error message. (The `-w` option modifies this behavior, as described in Table 1.4.) If the module is depended on by other modules, those modules are listed, so you can decide whether to unload them. If you want to unload an entire *module stack*—that is, a module and all those upon which it depends—you can use the `modprobe` command and its `-r` option, as described earlier in “Loading Kernel Modules.”

Summary

Before installing Linux, you should take some time to plan the implementation. This begins with determining which Linux distribution to use, and it continues with planning what installation media you want to use.

After installing Linux, you may need to attend to certain details. One of these is boot loader configuration. Although the installer usually gets this detail correct, particularly for single-OS systems, you may want to tweak the settings or add other OSs to the boot loader. You'll also need to understand this process when you install a new kernel down the road. In case you have problems booting your system, you should know how to troubleshoot boot problems by entering single-user mode, examining `dmesg` output, and using rescue discs.

Another common post-installation configuration detail is getting X working. Again, Linux distributions usually configure X correctly during installation, but you may need to tweak the settings or change them at a later date.

Finally, certain hardware management tools are vital to Linux administrators. You must be able to locate compatible hardware, identify the hardware you have installed, and configure it.

Exam Essentials

Summarize the concept of a Linux distribution. A distribution is a collection of software developed by diverse individuals and groups, bound by an installation routine. Linux distributions can differ in many details, but they all share the same heritage and the ability to run the same programs.

Describe when it's most appropriate to use CD-ROM and network installations. CD-ROM installations are most convenient when installing to systems with poor network connectivity or when you have a CD-ROM and want to install quickly. Network installations are convenient when you are installing several systems simultaneously or when you don't have a Linux CD-ROM or a CD-ROM drive on the target system.

Summarize the x86 boot process. The CPU executes code stored on the BIOS, which redirects the CPU to load and execute a boot loader from the MBR. This boot loader may load the OS kernel or redirect the boot process to another boot loader, which in turn loads the kernel and starts the OS running.

Explain the purpose of single-user mode. In single-user mode, Linux runs without most of the background processes or support for multiuser logins that are hallmarks of a normal Linux boot. Relieving the system of these processes enables you to perform low-level maintenance tasks that might not otherwise be possible.

Describe why you might use an emergency disc. An emergency disc enables you to boot a computer into Linux even when the Linux system installed on the computer is too badly damaged to boot. You can then use the disc-based system to perform data recovery or reconfigure the system on the hard disk.

Determine what video chipset your system uses. Many manufacturers document the video card chipset in their manuals or on the product boxes. You can also check the Microsoft Windows System Control Panel, if the manufacturer did not make the information readily available.

Summarize how X determines the monitor's refresh rate. X uses the monitor's maximum horizontal and vertical refresh rates and a series of fixed mode lines, which define particular timings for various video resolutions. X picks the mode line that produces the highest refresh rate supported by the monitor at the specified resolution.

Explain the purpose of an XDMCP server. XDM, KDM, and GDM are the three main Linux XDMCP servers. Each of them enables users to log into the system with X running, thus bypassing the text-mode login that many users find off-putting.

Summarize some tools for identifying hardware in Linux. The `lspci` and `lsusb` commands display information on PCI and USB hardware, respectively. The `lsmod` command displays the kernel driver modules that are loaded. The `/proc` filesystem is a treasure trove of hardware information, although many of its files contain highly technical information that can be difficult to interpret.

Review Questions

1. In what ways do Linux distributions differ from one another? (Choose all that apply.)
 - A. Package management systems
 - B. Kernel development history
 - C. Installation routines
 - D. The ability to run popular Unix servers
2. Which of the following best describes a typical Linux distribution's method of installation?
 - A. The installation program is a small Linux system that boots from floppy, CD-ROM, or hard disk to install a larger system on the hard disk.
 - B. The installation program is a set of DOS scripts that copies files to the hard disk, followed by a conversion program that turns the target partition into a Linux partition.
 - C. The installation program boots only from a network boot server to enable installation from CD-ROM or network connections.
 - D. The installation program runs under the Minix OS, which is small enough to fit on a floppy disk but can copy data to a Linux partition.
3. Which of the following is an advantage of a GUI installation over a text-based installation?
 - A. GUI installers support more hardware than do their text-based counterparts.
 - B. GUI installers can provide graphical representations of partition sizes, package browsers, and so on.
 - C. GUI installers can work even on video cards that support only VGA graphics.
 - D. GUI installers better test the system's hardware during the installation.
4. What is an advantage of a network installation over a DVD-ROM installation from a downloaded image file?
 - A. A network installation can result in less material downloaded.
 - B. A network installation will proceed more quickly once started.
 - C. A network installation will result in fewer disc swaps.
 - D. A network installation will work even if there's a DHCP error.
5. Where might the BIOS find a boot loader?
 - A. RAM
 - B. /dev/boot
 - C. MBR
 - D. /dev/kmem

6. Which command is used to install GRUB into the MBR of your first ATA hard drive?
 - A. **grub (hd0,1)**
 - B. **grub-install /dev/hda1**
 - C. **lilo /dev/hda**
 - D. **grub-install /dev/hda**
7. Which of the following files might you edit to configure GRUB? (Choose all that apply.)
 - A. **/boot/grub/menu.lst**
 - B. **/etc/grub.conf**
 - C. **/boot/grub/grub.conf**
 - D. **/boot/menu.conf**
8. The string **root (hd1,5)** appears in your **/boot/grub/menu.lst** file. What does this mean?
 - A. GRUB tells the kernel that its root partition is the fifth partition of the first disk.
 - B. GRUB looks for files on the sixth partition of the second disk.
 - C. GRUB looks for files on the fifth partition of the first disk.
 - D. GRUB installs itself in **/dev/hd1,5**.
9. What string would you add to the end of a GRUB **kernel** line to boot into single-user mode? (Choose all that apply.)
 - A. **1**
 - B. **single**
 - C. **emerg**
 - D. **one**
10. You want to examine the kernel ring buffer to debug a hardware problem. How would you do this?
 - A. Type **ringbuffer** at a command prompt.
 - B. Type **dmesg** at a command prompt.
 - C. Reboot and add the string **ring** to the kernel line in GRUB.
 - D. Install a Token Ring device and examine its output.
11. What is an advantage of using an emergency disc versus using single-user mode?
 - A. An emergency disc may work even if Linux won't boot into single-user mode.
 - B. An emergency disc may work even if you've lost the **root** password.
 - C. An emergency disc may work even if multiple users are logged in.
 - D. An emergency disc may work even if the CPU is defective.

12. To reset a `root` password that you've forgotten, you blank the second field in `root`'s entry in `/etc/shadow`. What should you do then?
 - A. Copy `/etc/shadow` so that you don't forget the password again.
 - B. Type **shadow-update** to update the shadow passwords.
 - C. Delete the entry for `root` in `/etc/passwd`.
 - D. Use `passwd` to set a new password for `root`.
13. Which of the following is the *most* useful information in locating an X driver for a video card?
 - A. The interrupt used by the video card under Microsoft Windows
 - B. The name of the driver used by the card under Microsoft Windows
 - C. Whether the card uses the ISA, VLB, PCI, or AGP bus
 - D. The name of the video card's manufacturer
14. Which of the following summarizes the organization of the `xorg.conf` file?
 - A. The file contains multiple sections, one for each screen. Each section includes subsections for individual components (keyboard, video card, and so on).
 - B. Configuration options are entered in any order desired. Options relating to specific components (keyboard, video card, and so on) may be interspersed.
 - C. The file begins with a summary of individual screens. Configuration options are preceded by a code word indicating the screen to which they apply.
 - D. The file is broken into sections, one or more for each component (keyboard, video card, and so on). The end of the file has one or more sections that define how to combine the main sections.
15. In what section of `xorg.conf` do you specify the resolution that you want to run?
 - A. In the `Screen` section, subsection `Display`, using the `Modes` option
 - B. In the `Monitor` section, using the `Modeline` option
 - C. In the `Device` section, using the `Modeline` option
 - D. In the `DefaultResolution` section, using the `Define` option
16. Which of the following features do KDM and GDM provide that XDM doesn't?
 - A. An encrypted remote X-based access ability, improving network security
 - B. The ability to accept logins from remote computers, once properly configured
 - C. The ability to select the login environment from a menu on the main login screen
 - D. A login screen that shows the username and password simultaneously rather than sequentially

17. You've installed a new PCI Ethernet card, but it doesn't seem to be working. What can you do to verify that the hardware is visible to Linux?
- A. Type **ping 127.0.0.1** to check connectivity.
 - B. Check that the Ethernet cable is firmly plugged in.
 - C. Type **cat /proc/ethernet** at a command prompt.
 - D. Type **lspci** at a command prompt.
18. An administrator types **lsusb** at a Linux command prompt. What type of information will appear?
- A. Basic identifying information on USB devices, including the USB controllers and all attached devices
 - B. Information on the drivers and detailed capabilities of all USB devices attached to the computer
 - C. A report on the success or failure to load the USB driver stack
 - D. A summary of the amount of data that's been transferred to and from USB devices since the computer was booted
19. Which of the following commands loads a kernel module? (Choose all that apply.)
- A. **rmmod**
 - B. **modprobe**
 - C. **lsmod**
 - D. **insmod**
20. You use a USB flash drive and, while the drive is still mounted, type **lsmod**, but you see no entries for kernel modules that you know are required to access a USB flash drive. Why might this be?
- A. The **lsmod** command displays information only on drivers that are in use by the user who typed the command.
 - B. The **lsmod** command displays information only on drivers that are doing work at the moment the command is typed.
 - C. The **lsmod** command displays information only on drivers that are built as modules, not those that are built directly into the kernel.
 - D. The **lsmod** command displays information only on drivers that are used by hardware devices internal to the computer.

Answers to Review Questions

1. A, C. Different Linux distributions use different package management systems and installation routines. Although they may ship with slightly different kernel versions, they use fundamentally the same kernel. Likewise, they may ship with different server collections but can run the same set of servers.
2. A. Most Linux distributions use installation programs written in Linux, not in DOS or Minix. The system usually boots from floppy or CD-ROM, although other boot media (such as hard disk or even network) are possible.
3. B. A bitmapped display, as used by a GUI installer, can be used to show graphical representations of the system's state that can't be done in a text-mode display. Text-based installers actually have an edge in hardware support because they can run on video cards that aren't supported by X.
4. A. When you download a DVD-ROM image file, you're almost certain to download programs you won't install, whereas with a direct network installation, the installer won't bother to download packages it doesn't install. Thus, option A is correct. Although it's conceivable that a network install will be faster than one from a DVD-ROM drive, this is not certain and probably isn't even likely; thus, option B is incorrect. Network installs and installs from a DVD-ROM are both likely to require no disc swaps, so option C is incorrect. A DHCP error refers to an inability to assign a computer an address on the network, so a network install might fail if this occurs, and option D is incorrect.
5. C. The master boot record (MBR) can contain a small boot loader. If more space is required, the boot loader must load a secondary boot loader. Although the boot loader is loaded into RAM, it's not stored there permanently because RAM is volatile storage. Both `/dev/boot` and `/dev/kmem` are references to files on Linux filesystems; they're meaningful only after the BIOS has found a boot loader and run it and lots of other boot processes have occurred.
6. D. You use `grub-install` to install the GRUB boot loader code into an MBR or boot sector. When using `grub-install`, you specify the boot sector on the command line. The MBR is the first sector on a hard drive, so you give it the Linux device identifier for the entire hard disk, `/dev/hda`. Option A specifies using the `grub` utility, which is an interactive tool, and the device identifier shown in option A is a GRUB-style identifier for what would probably be the `/dev/hda3` partition in Linux. Option B is almost correct but installs GRUB to the `/dev/hda1` partition's boot sector rather than the hard disk's MBR. Option C is the command to install LILO to the MBR rather than to install GRUB.
7. A, C. The official GRUB configuration filename is `/boot/grub/menu.lst`; however, some distributions use `/boot/grub/grub.conf` instead. Thus, options A and C are both correct. Options B and D are fictitious.

8. B. The `root` keyword in a GRUB configuration file tells GRUB where to look for files, including its own configuration files, kernel files, and so on. Because GRUB numbers start from 0, `(hd1, 5)` refers to the sixth partition on the second disk. Option A is incorrect because you pass the Linux root partition to the kernel on the kernel line. Options A and C both misinterpret the GRUB numbering scheme. The GRUB installation location is specified on the `grub-install` command line, and `/dev/hd1, 5` isn't a standard Linux device file.
9. A, B. Linux recognizes both `1` and `single` as codes to enter single-user mode in this context. Neither `emerg` nor `one` is a correct answer.
10. B. The `dmesg` command displays the contents of the kernel ring buffer, which holds kernel messages. There is no standard `ringbuffer` command. Adding `ring` to the kernel options when booting will not have the desired effect. Token Ring is a network hardware standard, not a way to examine the kernel ring buffer.
11. A. If Linux is so badly damaged that it won't boot into single-user mode, there's a chance that an emergency disc will give you sufficient access to the computer to fix the problem. Single-user mode bypasses the usual login procedures, so it can work even if you've lost the root password, so option B is incorrect. Both emergency discs and single-user mode require ordinary users to log out, so option C is incorrect. If the CPU is defective, as option D specifies, chances are that neither single-user mode nor an emergency disc will help.
12. D. Blanking the password field as specified in the question results in a null password—no password is required to access the account. This is extremely dangerous, so you should immediately set a new password for the `root` account using `passwd`, as option D specifies. Copying `/etc/shadow` is not required and might pose a security risk, depending on where and how you copy it. There is no standard `shadow-update` utility. Deleting `root` from `/etc/passwd` would likely have disastrous consequences, since this file holds basic account information.
13. B. The driver used under Windows should provide useful information on the video card's chipset, which will help you locate a Linux driver for the card. The video card's manufacturer name might or might not be useful information. If it proves to be useful, you'd also need a model number. The interrupt used by the video card in Windows is irrelevant. The card's bus can narrow the range of possibilities, but it isn't extremely helpful.
14. D. The `xorg.conf` file design enables you to define variants or multiple components and easily combine or recombine them as necessary.
15. A. The `Modeline` option in the Monitor section defines *one* possible resolution, but there are usually several `Modeline` entries defining many resolutions. The `Modeline` option doesn't exist in the Device section, however, nor is that section where the resolution is set. There is no `DefaultResolution` section.
16. C. KDM and GDM add many features, one of which is a menu that enables users to select their desktop environment or window manager when they log in rather than specifying it in a configuration file, as option C states. Option A describes one of the advantages of the Secure Shell (SSH) as a remote-access protocol. Option B describes a feature common to all three XDMCP servers. Option D describes the way both KDM and XDM function; GDM is the one that presents username and password fields in series rather than simultaneously.

17. D. The `lspci` command lists all the detected PCI devices, so if the card is installed correctly and working, it should show up in the `lspci` output. Thus, option D is correct. Although `ping` can be a useful network diagnostic tool, option A's use of it doesn't test Ethernet cards, so that option is incorrect. Option B provides sound basic network debugging advice, but it won't help to verify that the card is visible to Linux. There is no `/proc/ethernet` file, so option C won't help.
18. A. The `lsusb` command summarizes the USB devices that are available to the computer, as option A says.
19. B, D. The `modprobe` command loads a module and, if necessary, all those upon which it depends. The `insmod` command loads a module, but only if all its dependencies are met. Thus, options B and D are correct. The `rmmmod` command removes a module, and `lsmod` lists the loaded modules, so options A and C are incorrect.
20. C. If a driver is built into the main kernel file, `lsmod` won't display information on it. Thus, if the relevant drivers are built into the kernel, the observed behavior would occur because option C is a correct statement. The `lsmod` command does display information on drivers that are used to service other users' needs, that are loaded but not actively working, and on some types of external hardware devices, contrary to options A, B, and D, respectively.

Chapter 2

Using Text-Mode Commands

THE FOLLOWING COMPTIA OBJECTIVES ARE COVERED IN THIS CHAPTER:

- ✓ **1.9 Configure profile and environment variables system-wide and at the user level (PS1, PS2, PATH, EDITOR, TERM, PAGER, HOME, PRINTER).**
- ✓ **2.1 Given a scenario, use the following fundamental Linux tools, techniques, and resources (Directory navigation: `cd`, `ls`, `pushd`, `popd`, `pwd`; File commands: `file`, `test`, `find`, `locate`, `slocate`, `which`, `whereis`, `ln`, `ls -F`, `mknod`, `touch`, `mkdir`, `mv`, `cp`, `rm`, `cd`; file types [hard links, soft links, directory, device file, regular file, named pipe; I/O redirection: `<`, `>`, `=`, `==`, `|`, `;`, `tee`, `xargs`, **STDIN**, **STDOUT**, **STDERR**; Special devices: `/dev/null`, `/dev/random`, `/dev/zero`, `/dev/urandom`; System documentation: **Man pages** [`man#`, `apropos`, `makewhatis`, `whatis`], **Info pages**, `/usr/share/docs`)**
- ✓ **2.2 Conduct basic tasks using BASH (Basics of scripting [only: execute permission, `#!/bin/bash`, `sh script`]; Shell features: history, tab completion; Special devices: `/dev/null`, `/dev/random`, `/dev/zero`, `/dev/urandom`).**
- ✓ **5.2 Given a scenario, select the appropriate file permissions and ownership and troubleshoot common problems (Tools: `chmod`, `chown`, `chroot`, `chgrp`, `lsattr`, `chattr`, `umask`. Special permissions: `setuid`, `setgid`, sticky bit).**



Linux can trace its intellectual heritage, if not its source code, to the Unix OS. Unix was developed before GUI environments were much more than pipe dreams. Thus, Unix (and hence

Linux) provides a wide array of flexible text-mode commands. In fact, even many GUI tools are built atop the text-mode commands—the GUI tools simply translate mouse clicks into options passed to the text-mode tools and display any output in a flashier way than the originals. In any event, because of Linux’s strong text-mode heritage, Linux administrators, and even some nonadministrative Linux users, must understand how to use these text-mode tools. This chapter serves as an introduction to this topic.

The most fundamental text-mode tool is a command shell, which accepts typed commands from a user. Thus, this chapter begins with a look at shells. It then moves on to a look at many commands that are used to manipulate files in various ways—to display their contents, move them, and so on. One of the features of files is that they have access controls (that is, permissions), and understanding these permissions and the commands to manipulate them is critical for many Linux tasks, so this chapter covers this important topic. Linux also provides various tools for manipulating text files, so that topic is also covered in this chapter. Many commands rely on environment variables, which store small amounts of data that can be used by multiple commands, so knowing how to set environment variables can be important. This chapter continues with a basic examination of scripts, which enable the automation of common tasks. This chapter concludes with information on system documentation and help resources, so you can get help on using a command if you forget a critical detail.



Objective 2.1 is covered partly in this chapter and partly in Chapter 3.

Using a Command Shell

A *shell* is a program that enables you to interact with the computer by launching programs, manipulating files, and issuing commands. A shell is sometimes referred to as a *command-line interface (CLI)*. Shells aren’t the same as the GUI desktop environments with which you may already be familiar, though; traditional Linux shells are text-mode tools. Even if you prefer to use a GUI environment, it’s important that you understand basic shell use because the shell provides the user interface that’s most consistent across distributions and

other environments. You can also use text-based shells through text-mode network connections. Once you've started a shell, you can view and manipulate files and launch programs.

Starting a Shell

Linux supports many different shells, although precisely which ones might be installed varies from one distribution to another. The vast majority of Linux systems include Bash, which is usually the default shell for new users. Another common shell is known as `tcsh`, and many others, such as `zsh`, `csh`, and `ash`, are also available. Most shells are similar in broad strokes, but some details differ.

You can start a shell in many different ways, most of which are at least partially automatic. The following are the most common methods:

Logging in at the text-mode console If you log into the computer using a text-mode console, you'll be greeted by your default shell, as it is set in your user account information (see Chapter 5, "Managing Users").

Logging in remotely Logging in remotely via Telnet, the Secure Shell (SSH), or some other remote text-mode login tool will start a shell. (Despite its name, SSH is not a shell in the sense described here, but it will start one automatically.)

Starting an *xterm* An *xterm* is a GUI program in which text-based programs can run. By default, an *xterm* usually starts your default shell unless told to do otherwise.

Explicitly launching a shell You can start one shell from within another. This can be helpful if you find you need features of one shell but are running another. Type the new shell's name to start it.

When you start a shell, you'll see a *command prompt*. This is one or more characters that indicate the shell is waiting for input. Command prompts often (but not always) include your username, the computer's hostname, or the directory in which the shell is operating. For instance, a command prompt might resemble the following:

```
[rodsmith@nessus /mnt]$
```

Although not a universal convention (it can be set in a user's shell configuration files), the final character is often a dollar sign (\$) or greater-than symbol (>) for ordinary users or a hash mark (#) for `root`. This serves as an indication of superuser status; you should be cautious when entering commands in a `root` shell, because it's easy to damage the system from such a shell.



This book includes command examples on separate lines. When the command is one that an ordinary user might issue, it's preceded by a \$ prompt; when only `root` should be issuing the command, it's preceded by a # prompt. Because the username, computer name, and directory are usually unimportant, this information is omitted from the prompts printed in this book. The prompts are omitted from command examples within a paragraph of text.

Using Virtual Terminals

Linux implements a feature, known as *virtual terminals* (VTs), that can greatly help you when you want to run multiple programs from a text-mode login or when you want to switch between text-mode and GUI sessions. A single computer can support multiple VTs. You can switch to a new VT from a text-mode login by pressing **Alt+F n** , where n is the VT number. For instance, if you're in VT1 and you want to switch to VT2, press **Alt+F2**. To switch out of a VT in which X is running, you must add **Ctrl** to the keystroke; so, to switch to VT2 from X, you'd press **Ctrl+Alt+F2**.

Typically, VT1 through VT6 are text-mode VTs, while X runs in VT7. Some distributions deviate from this convention, though. For instance, Fedora 10 (but not earlier versions of Fedora) run X in VT1.

VTs enable you to log into the computer multiple times and run different programs in different VTs. You can then switch between the VTs to switch between different programs running in each VT.



X sometimes misbehaves and becomes unresponsive. If this happens, you may be able to recover by switching to a text-mode VT and shutting down and restarting X, as described in Chapter 1, “Getting Started with Linux.”

Launching Programs

You can launch a program from a shell by typing its name. In fact, many shell “commands” are actually external programs that the shell runs. Most of these standard commands reside in the `/bin` directory, but shells search all directories specified by the `PATH` environment variable (described in more detail later, in “Setting Environment Variables”) for commands to run. If you type the name of a program that resides in any directory on the path, the shell runs that program. You can also pass *parameters* to a program—optional information that the program can use in a program-specific way. For instance, the names of the files and directories are parameters to commands like `ls` and `cd`, which are described later, in “Navigating the Linux Filesystem.” Many programs accept parameters that are preceded by one or two dashes and a code, as in `-r` or `-t` *time*. Most parameters are case sensitive; in fact, many programs use upper- and lowercase versions of a parameter in different ways.

Most text-based programs take over the display (the text-mode login, Telnet session, `xterm`, or what have you). Many show little or no information before returning control to the shell, so you don't really notice this fact. Some programs, such as text-mode editors, truly control the display; they may clear all the information that has previously appeared and fill the display with their own information. Other programs may not clear the screen entirely, or even display their own information, but they may take a long time to operate. In some cases, you may want to retain control of your shell while the program does its thing in the background. To do this, follow the command with an ampersand (`&`). When you do this, the program you launch will still be attached to the display from which it

was launched, but it shares that display with the shell. This works well for noninteractive programs but very poorly for interactive tools. For instance, suppose you have a program called `supercrunch` that performs some lengthy computation but requires no interaction from the user. You could launch it like this:

```
$ supercrunch &
```

If `supercrunch` produces text-based output, it will appear on the screen, but you'll still be able to use the shell for other purposes. If you've already launched a program and want to move it into the background, press `Ctrl+Z`. This suspends the currently running program and returns you to the shell. At this point, the program you've suspended will *not* be doing any work. This may be fine for a text editor you wanted to momentarily suspend, but if the program was performing computations that should continue, you must take additional steps to see that this happens. You can type **fg** to return to the suspended program or type **bg** to start it running again in the background. The latter is much like appending an ampersand to the command name when you launched it.

If you try to launch an X-based program, you must be running the shell in an `xterm`, or possibly in some other way that allows X programs to run, such as from another computer with its own X server and all appropriate environment variables set to permit remote X program operation, as described in Chapter 9, "Configuring Advanced Networking." If you try to launch an X program from a text-only login, you'll receive an error message along the lines of `Can't open display`.



Although X-based programs don't normally produce text output, they do take over the terminal from which they were launched. If you want to continue to use a terminal after launching an X-based program, follow its name with an ampersand (&), as just described.

Using Shell Shortcuts

Linux shells permit some important shortcuts. One of these is the use of the `Tab` key for *filename completion*. Suppose you want to move a file that's called `shareholder-report-for-2009.txt`. You could type the entire filename, but that can be tedious. Most Linux shells, including the popular `Bash`, support a feature in which hitting the `Tab` key completes an incomplete command or filename, as long as you've typed enough characters to uniquely define the file or command. For instance, suppose that `ls` (described in more detail shortly, in "Listing Files") reveals two files in a directory:

```
$ ls
share-price-in-2009.txt  shareholder-report-for-2009.txt
```

If you want to edit the second file with the Emacs editor (using the command name `emacs`), you could type **emacs shareh** and then press the `Tab` key. The shell will complete the filename.

What happens when the characters you enter are *not* unique? In this case, the shell completes as much of the job as it can. For instance, if you type **emacs sh** and then press the Tab key, Bash fills out the next three characters so that the command line reads **emacs share**. Some configurations also summarize the possible completions at this point. (For those that don't, pressing Tab again usually displays these completions.) If you then type either **h** or **-** and press Tab again, Bash completes the filename.



Command and filename completion details vary; you may need to press Tab multiple times and there may be fewer or more beeps than described here.

Another shortcut is the use of the up and down arrow keys to scroll through previous commands. If you need to type two similar commands in a row, you can type one and then press the up arrow key to retrieve the previous command. You can go back through several commands in this way, and if you overshoot, you can use the down arrow key to retrieve more recent commands. Once you find the command you want, you can use the left arrow or Backspace key to move back in the line to edit it (Backspace deletes characters, but the left arrow key doesn't). Pressing Ctrl+A moves the cursor to the start of the line, and pressing Ctrl+E moves the cursor to the end of the line. Edit the line, and press the Enter key to enter the new command.

These shortcuts, and other basic shell commands, are extremely helpful. You can perform many tasks with a file manager, of course, but text-based utilities were designed to be used from shells.

Using the Shell's History

Another helpful shell shortcut is the *history*, which keeps a record of every command you type (stored in `~/.bash_history` in the case of Bash), up to a configurable number of commands. If you've typed a long command recently and want to use it again, or use a minor variant of it, you can pull the command out of the history. The simplest way to do this is to press the up and down arrow keys on your keyboard, as described in the previous section. The Ctrl+P and Ctrl+N keystrokes double for the up and down arrow keys, respectively.

Another way to use the command history is to search through it. Press Ctrl+R to begin a backward (reverse) search, and begin typing characters that should be unique to the command you want to find. The characters you type need not be the ones that begin the command; they can exist anywhere in the command. You can either keep typing until you find the correct command or, after you've typed a few characters, press Ctrl+R repeatedly until you find the one you want. The Ctrl+S keystroke works similarly but searches forward in the command history, which might be handy if you've used a backward search or the up arrow key to look back and have overshoot. In either event, if you can't find the command you want or change your mind and want to terminate the search, press Ctrl+G.

Frequently, after finding a command in the history, you want to edit it. Bash, like many shells, provides editing features modeled after those of the Emacs editor:

Move within the line. Press Ctrl+A or Ctrl+E to move the cursor to the start or end of the line, respectively. The left and right arrow keys move within the line a character at a time. Ctrl+B and Ctrl+F do the same, moving backward and forward within a line. Pressing Ctrl plus the left or right arrow key moves backward or forward a word at a time, as does pressing Esc and then B or F.

Delete text. Pressing Ctrl+D or the Delete key deletes the character under the cursor, whereas pressing the Backspace key deletes the character to the left of the cursor. Pressing Ctrl+K deletes all text from the cursor to the end of the line. Pressing Ctrl+X and then Backspace deletes all the text from the cursor to the beginning of the line.

Transpose text. Pressing Ctrl+T transposes the character before the cursor with the character under the cursor. Pressing Esc and then T transposes the two words immediately before (or under) the cursor.

Change case. Pressing Esc and then U converts text from the cursor to the end of the word to uppercase. Pressing Esc and then L converts text from the cursor to the end of the word to lowercase. Pressing Esc and then C converts the letter under the cursor (or the first letter of the next word) to uppercase, leaving the rest of the word unaffected.

Invoke an editor. You can launch a full-fledged editor to edit a command by pressing Ctrl+X followed by Ctrl+E. Bash attempts to launch the editor defined by the FCEDIT or EDITOR environment variable or Emacs as a last resort. (The upcoming section “Setting Environment Variables” describes environment variables.)

These editing commands are just the most useful ones supported by Bash; consult its man page to learn about many more obscure editing features. In practice, you’re likely to make heavy use of command and filename completion, the command history, and perhaps a few editing features.

The `history` command provides an interface to view and manage the history. Typing **history** alone displays all the commands in the history (typically the latest 500 commands); adding a number causes only that number of the latest commands to appear. Typing **history -c** clears the history, which can be handy if you’ve recently typed commands you’d rather not have discovered by others (such as commands that include passwords). You can rerun a specific command by typing its number preceded by an exclamation mark (!), as in **!227** to rerun the command that’s numbered 227 in the list.

Manipulating Files and Directories

Linux provides traditional Unix commands to manipulate files. These commands can be classified into several categories: filesystem navigation, file manipulation, directory manipulation, file location, and file examination. A couple of closely related features are redirection and pipes, which let you redirect a program’s input or output from or to a file or another program.

Navigating the Linux Filesystem

Moving about the Linux filesystem involves a few commands. It's also helpful to understand some features of common Linux shells that can help in this navigation. Important tasks include taking directory listings, using wildcards, and manipulating the current directory.

Listing Files

To manipulate files, it's helpful to know what they are. This is the job of the `ls` command, whose name is short for “list.” The `ls` command displays the names of files in a directory. Its syntax is simple:

```
ls [options] [files]
```

The command supports a huge number of options, which are documented in its man page. (See the upcoming section “Getting Help” for information on man pages.) Table 2.1 summarizes the most useful options.

TABLE 2.1 Common `ls` Options

Option Name	Option Abbreviation	Meaning
<code>--all</code>	<code>-a</code>	Normally, <code>ls</code> omits files whose names begin with a dot (.). These dot files are often configuration files that aren't usually of interest. Adding the <code>-a</code> or <code>--all</code> parameter displays dot files.
<code>--color</code>	None	This option produces a color-coded listing that differentiates directories, symbolic links, and so on, by displaying them in different colors. (A few types of displays don't support color, though.)
<code>--directory</code>	<code>-d</code>	Normally, if you type a directory name as one of the <i>files</i> , <code>ls</code> displays the contents of that directory. The same thing happens if a directory name matches a wildcard (described in the next section, “Using Wildcards”). Adding the <code>-d</code> or <code>--directory</code> parameter changes this behavior to list only the directory name, which is sometimes preferable.
None	<code>-l</code>	The <code>ls</code> command normally displays filenames only. The <code>-l</code> parameter (a lowercase <i>L</i>) produces a long listing that includes information such as the file's permission string (described later, in “Using File Permissions”), owner, group, size, and creation date.

TABLE 2.1 Common ls Options (continued)

Option Name	Option Abbreviation	Meaning
--classify	-F	This option appends an indicator code to the end of each name so you know what type of file it is. The meanings are as follows: / directory @ symbolic link = socket pipe
--recursive	-R	This option causes ls to display directory contents recursively. That is, if the target directory contains a subdirectory, ls displays both the files in the target directory <i>and</i> the files in its subdirectory. The result can be a huge listing if a directory has many subdirectories.

Both the *options* list and the *files* list are optional. If you omit the *files* list, ls displays the contents of the current directory. You may instead give one or more file or directory names, in which case ls displays information on those files or directories; for instance:

```
$ ls -F /usr /bin/ls
/bin/ls

/usr:
X11R6/  games/          include/  man/      src/
bin/    i386-glibc20-linux/ lib/      merge@   tmp@
doc/    i486-linux-libc5/  libexec/  sbin/
etc/    i586-mandrake-linux/ local/    share/
```

This output shows both the /bin/ls program file and the contents of the /usr directory. The latter consists mainly of subdirectories, but it includes a couple of symbolic links as well. By default, ls creates a listing that’s sorted by filename, as shown in this example. Note, though, that uppercase letters (as in X11R6) always appear before lowercase letters (as in bin).



Linux uses a slash (/) to separate elements of a directory. Windows uses a backslash (\) for this purpose, and Mac OS Classic uses a colon (:). (Mac OS X is Unix-based and uses a slash, just like Linux.)

One of the most common `ls` options is `-l`, which creates a listing like this:

```
$ ls -l n*
-rw-r--r-- 1 rodsmith users 198629 2008-12-23 13:25 ndiswrapper-1.53.tar.gz
-rw-r--r-- 1 rodsmith users 1138860 2009-03-13 11:09 nedit-5.5-1.i386.rpm
-rw-r--r-- 1 rodsmith users 113747 2009-06-13 22:11 ntp.pdf
```

This output includes the permission strings, ownership, file sizes, and file creation dates in addition to the filenames. This example also illustrates the use of the `*` wildcard, which matches any string—thus, `n*` matches any filename that begins with `n`.

Using Wildcards

You can use *wildcards* with `ls` (and with many other commands as well). A wildcard is a symbol or set of symbols that stand in for other characters. Three classes of wildcards are common in Linux:

? A question mark (?) stands in for a single character. For instance, `b??k` matches `book`, `ba1k`, `buck`, or any other four-letter filename that begins with `b` and ends with `k`.

***** An asterisk (*) matches any character or set of characters, including no character. For instance, `b*k` matches `book`, `ba1k`, and `buck`, just as does `b??k`. `b*k` also matches `bk`, `bbk`, and `backtrack`.

Bracketed values Characters enclosed in square brackets ([]) normally match any character in the set. For instance, `b[ao][lo]k` matches `ba1k` and `book` but not `buck`. It's also possible to specify a range of values; for instance, `b[a-z]ck` matches any `back`, `buck`, and other four-letter filenames of this form whose second character is a lowercase letter. This differs from `b?ck`—because Linux treats filenames in a case-sensitive way, `b[a-z]ck` doesn't match `bAck`, although `b?ck` does.

Wildcards are actually implemented in the shell and passed to the command you call. For instance, if you type `ls b??k` and that wildcard matches the three files `ba1k`, `book`, and `buck`, the result is precisely as if you'd typed `ls ba1k book buck`.



The way wildcards are expanded can lead to some undesirable consequences. For instance, suppose you want to copy two files, specified via a wildcard, to another directory but you forget to give the destination directory. The `cp` command (described shortly) will interpret the command as a request to copy one of the files over the other.

Finding and Changing the Current Directory

Linux command shells implement the concept of a *current directory*, a directory that's displayed by default if `ls` or some other command doesn't specify a directory. You can discover what your current directory is by typing `pwd`. This command's name stands for "print working directory," and it can be useful if you don't know in what directory you're currently operating.

You may specify either an *absolute directory name* or a *relative directory name* when giving a filename or directory name. The former indicates the directory name relative to the root directory. An absolute directory name uses a leading slash, as in `/usr/local` or `/home`. Relative directory names are specified relative to the current directory. They lack the leading slash. Relative directory names sometimes begin with a double dot (`..`). This is a code that stands for a directory's parent. For instance, if your current directory is `/usr/local`, then `..` refers to `/usr`. Similarly, a single dot (`.`) as a directory name refers to the current directory. As an example, if you're in `/home/sally`, the filename specifications `document.odt`, `./document.odt`, and `/home/sally/document.odt` all refer to the same file. The single dot can often be omitted, but including it is sometimes helpful when you're specifying commands. Without the dot, Linux tries searching your path, and if the dot isn't on the path and you aren't in a directory on the path, you won't be able to run programs in your current working directory.

Another important shortcut character is the tilde (`~`). This character is a stand-in for your home directory. For instance, `~/document.odt` refers to the `document.odt` file within the user's home directory. This might be `/home/sally/document.odt` for the user `sally`, for instance.

To change to another directory, use the `cd` command. Unlike most commands, `cd` is built into the shell. Its name stands for “change directory,” and it alters the current directory to whatever you specify. Type the command followed by your target directory, as in:

```
$ cd somedir
```

You may use either absolute or relative directory names with the `cd` command—or with other commands that take filenames or directory names as input.

An alternative to `cd` is `pushd`, which changes to another directory and adds it to a list of directories maintained by the shell. You can view the list with the `dirs` command (the list is also displayed after every call to `pushd`), and you can also rotate the list and therefore change to a specified directory by passing a number to the list, indexed from 0, with positive numbers counting from the left and negative numbers counting from the right. For instance:

```
$ pushd /tmp
/tmp ~
$ pushd /usr/local
/usr/local /tmp ~
$ pushd +2
~ /usr/local /tmp
$ pwd
/home/rodsmith
```

To remove a directory from the list, use `popd`, which removes the first directory from the list or the numbered directory (using positive or negative values, as in `pushd`):

```
$ dirs
~ /usr/local /tmp
```

```
$ popd -0
~ /usr/local
```

Manipulating Files

A few file-manipulation commands are extremely important to everyday file operations. These commands enable you to copy, move, rename, and delete files.

Copying Files

The `cp` command copies a file. Its basic syntax is as follows:

```
cp [options] source destination
```

source is normally one or more files, and *destination* may be a file (when the source is a single file) or a directory (when the source is one or more files). When copying to a directory, `cp` preserves the original filename; otherwise, it gives the new file the filename indicated by *destination*. The command supports a large number of options; consult its man page for more information. (The upcoming section “Getting Help” describes man pages.) Some of the more useful options enable you to modify the command’s operation in helpful ways, as summarized in Table 2.2.

TABLE 2.2 Common `cp` Options

Option Name	Option Abbreviation	Meaning
<code>--force</code>	<code>-f</code>	This option forces the system to overwrite any existing files without prompting.
<code>--interactive</code>	<code>-i</code>	This option causes <code>cp</code> to ask you before overwriting any existing files.
<code>--preserve</code>	<code>-p</code>	Normally, a copied file is owned by the user who issues the <code>cp</code> command and uses that account’s default permissions. This option preserves ownership and permissions, if possible.
<code>--recursive</code>	<code>-R</code> (and sometimes <code>-r</code>)	If you use this option and specify a directory as <i>source</i> , the entire directory, including its subdirectories, will be copied. Although <code>-r</code> also performs a recursive copy, its behavior with files other than ordinary files and directories is unspecified. Most <code>cp</code> implementation use <code>-r</code> as a synonym for <code>-R</code> , but this isn’t guaranteed.
<code>--update</code>	<code>-u</code>	This option tells <code>cp</code> to copy the file only if the original is newer than the target or if the target doesn’t exist.

As an example, the following command copies the `/etc/fstab` configuration file to a backup location in `/root`, but only if the original `/etc/fstab` is newer than the existing backup:

```
# cp -u /etc/fstab /root/fstab-backup
```

Moving and Renaming Files

The `mv` command (short for “move”) is commonly used both to move files and directories from one location to another and to rename them. Linux doesn’t distinguish between these two types of operations, although many users do. The syntax of `mv` is similar to that of `cp`:

```
mv [options] source destination
```

The command takes many of the same *options* as `cp` does. From Table 2.2, `--preserve` and `--recursive` don’t apply to `mv`, but the others do.

To move one or more files or directories, specify the files as *source* and specify a directory or (optionally for a single file move) a filename for *destination*:

```
$ mv document.odt important/purchases/
```

This command copies the `document.odt` file into the `important/purchases` subdirectory. If the copy occurs on one low-level filesystem, Linux does the job by rewriting directory entries; the file itself doesn’t need to be read and rewritten. This makes `mv` fast. When the target directory is on another partition or disk, though, Linux must read the original file, rewrite it to the new location, and delete the original. This slows down `mv`. Also, `mv` can move entire directories within a filesystem, but not between filesystems.



The preceding example used a trailing slash (/) on the destination directory. This practice can help avoid problems caused by typos. For instance, if the destination directory were mistyped as `important/purchase` (missing the final `s`), `mv` would move `document.odt` into the `important` directory under the filename `purchase`. Adding the trailing slash makes it explicit that you intend to move the file into a subdirectory. If it doesn’t exist, `mv` complains, so you’re not left with mysterious misnamed files. You can also use filename completion to avoid such problems.

Renaming a file with `mv` works much like moving a file, except that the source and destination filenames are in the same directory, as in:

```
$ mv document.odt washer-order.odt
```

This renames `document.odt` to `washer-order.odt` in the same directory. You can combine these two forms as well:

```
$ mv document.odt important/purchases/washer-order.odt
```

This command simultaneously moves and renames the file.

Removing Files

To delete a file, use the `rm` command, whose name is short for “remove.” Its syntax is simple:

```
rm [options] files
```

The `rm` command accepts many of the same *options* as `cp` or `mv`. Of those described in Table 2.2, `--preserve` and `--update` do not apply to `rm`, but all the others do. With `rm`, `-r` is synonymous with `-R`.



By default, Linux doesn't provide any sort of “trash-can” functionality for its `rm` command; once you've deleted a file with `rm`, it's gone and cannot be recovered without retrieving it from a backup or performing low-level disk maintenance. Therefore, you should be cautious when using `rm`, especially as root. This is particularly true when you're using the `-R` option—typing **`rm -R /`** will destroy an entire Linux installation! Many Linux GUI file managers implement trash-can functionality so that you can easily recover files moved to the trash (assuming you haven't emptied the trash), so you may want to use a file manager for removing files.

Touching Files

The `touch` command has two purposes: it can create a new empty file or it can adjust the last-modification and last-access times of an existing file. Using `touch` in the first way is helpful if you need to create an empty file for some reason (say, if another program expects a file to exist, but that file doesn't need to have any content). Changing the file's timestamp is useful if a program relies on this information and you need to change it. Programmers sometimes employ this feature to force development tools to recompile a source code file even if it hasn't been modified since it was last compiled, for instance.

Creating Links

The `ln` command creates *hard links* and *soft links* (aka *symbolic links*). Its syntax is similar to that of `cp`:

```
ln [options] source link
```

source is the original file, while *link* is the name of the link you want to create. This command supports options that have several effects, as summarized in Table 2.3.

TABLE 2.3 Common ln Options

Option Name	Option Abbreviation	Meaning
--force	-f	This option causes ln to remove any existing links or files that have the target <i>link</i> name.
--directory	-d or -F	Ordinarily, you can't create hard links to directories. The root user can attempt to do so, though, by passing this option to ln. (Symbolic links to directories are not a problem. This distinction is described shortly.) In practice, this option is unlikely to work, since few filesystems support hard links to directories.
--symbolic	-s	The ln command creates hard links by default. To create a symbolic link, pass this option to the command.

A few other options exist to perform more obscure tasks; consult the ln man page for details. The default type of link created by ln, hard links, are produced by creating two directory entries that point to the same file. Both filenames are equally valid and prominent; neither is a “truer” filename than the other, except that one was created first (when creating the file) and the other was created second. To delete the file, you must delete both hard links to the file. Because of the way hard links are created, all the links to a single file must exist on one low-level filesystem; you can't create a hard link from, say, your root (/) filesystem to a separate filesystem you've mounted on it, such as your /home filesystem (if it's on a separate partition). The underlying filesystem must support hard links. All Linux native filesystems support this feature, but some non-Linux filesystems don't.

Symbolic links, by contrast, are special file types. The symbolic link is a separate file whose contents point to the linked-to file. Linux knows to access the linked-to file when you try to access the symbolic link, so in most respects accessing a symbolic link works just like accessing the original file. Because symbolic links are basically files that contain filenames, they can point across low-level filesystems—you can point from the root (/) filesystem to a file on a separate /home filesystem, for instance. The lookup process for accessing the original file from the link consumes a tiny bit of time, so symbolic link access is slower than hard link access—but not by enough that you'd notice in any but very bizarre conditions or artificial tests. Long directory listings show the linked-to file:

```
$ ls -l alink.odt
lrwxrwxrwx 1 rodsmith users 8 Dec  2 15:31 alink.odt -> test.odt
```

Manipulating Directories

Files normally reside in directories. Even normal users frequently create, delete, and otherwise manipulate directories. Some of the preceding commands can be used with directories—you can move or rename directories with `mv`, for instance. The `rm` command won't delete a directory unless used in conjunction with the `-R` parameter. Linux provides additional commands for manipulating directories.

Creating Directories

The `mkdir` command creates (or makes, which is the reason for the command name) a directory. This command's official syntax is as follows:

```
mkdir [options] directory-names
```

In most cases, `mkdir` is used without *options*, but a few are supported, as summarized in Table 2.4.

TABLE 2.4 Common `mkdir` Options

Option Name	Option Abbreviation	Meaning
<code>--mode=mode</code>	<code>-m mode</code>	This option causes the new directory to have the specified permission mode, expressed as an octal number. (The upcoming section "Using File Permissions" describes permission modes.)
<code>--parents</code>	<code>-p</code>	Normally, if you specify the creation of a directory within another directory that doesn't exist, <code>mkdir</code> responds with a <code>No such file or directory</code> error and doesn't create the directory. If you include this option, though, <code>mkdir</code> creates the necessary parent directory.

Removing Directories

The `rmdir` command is the opposite of `mkdir`; it destroys a directory. Its syntax is similar:

```
rmdir [options] directory-names
```

Like `mkdir`, `rmdir` supports few options, the most important of which are described in Table 2.5.

TABLE 2.5 Common `rmdir` Options

Option Name	Option Abbreviation	Meaning
<code>--ignore-fail-on-non-empty</code>	None	Normally, if a directory contains files or other directories, <code>rmdir</code> won't delete it and returns an error message. With this option, <code>rmdir</code> still won't delete the directory, but it doesn't return an error message.
<code>--parents</code>	<code>-p</code>	This option causes <code>rmdir</code> to delete an entire directory tree. For instance, typing <code>rmdir -p one/two/three</code> causes <code>rmdir</code> to delete <code>one/two/three</code> , then <code>one/two</code> , and finally <code>one</code> , provided no other files or directories are present.



When you're deleting an entire directory tree filled with files, `rm -R` is a better choice than `rmdir` because `rm -R` deletes files within the specified directory but `rmdir` doesn't.

Locating Files

You use file location commands to locate a file on your computer. Most frequently, these commands help you locate a file by name or sometimes by other criteria, such as modification date. These commands can search a directory tree (including root, which scans the entire system) for a file matching the specified criteria in any subdirectory.

Using the *find* Command

The `find` utility implements a brute-force approach to finding files. This program finds files by searching through the specified directory tree, checking filenames, file creation dates, and so on, to locate the files that match the specified criteria. Because of this method of operation, `find` tends to be slow, but it's very flexible and is very likely to succeed, assuming the file for which you're searching exists. The `find` syntax is as follows:

```
find [path...] [expression...]
```

You can specify one or more paths in which `find` should operate; the program will restrict its operations to these paths. *expression* is a way of specifying what you want to find. The `find` man page includes information on these expressions, but some of the more common enable you to search by popular criteria, as summarized in Table 2.6.

TABLE 2.6 Common `find` Expressions

Expression Name	Meaning
<code>--name <i>pattern</i></code>	You can search for a filename using this expression. Doing so finds files that match the specified <i>pattern</i> . If <i>pattern</i> is an ordinary filename, <code>find</code> matches that name exactly. You can use wildcards if you enclose <i>pattern</i> in quotes, and <code>find</code> will locate files that match the wildcard filename.
<code>--perm <i>mode</i></code>	If you need to find files that have certain permissions, you can do so by using this expression. The <i>mode</i> may be expressed either symbolically or in octal form. If you precede <i>mode</i> with a <code>+</code> , <code>find</code> locates files in which <i>any</i> of the specified permission bits are set. If you precede <i>mode</i> with a <code>-</code> , <code>find</code> locates files in which <i>all</i> the specified permission bits are set. (The upcoming section “Using File Permissions” describes file modes.)
<code>--size <i>n</i></code>	You can search for a file of a given size with this expression. Normally, <i>n</i> is specified in 512-byte blocks, but you can modify this by trailing the value with a letter code, such as <code>c</code> for bytes or <code>k</code> for kilobytes.
<code>--gid <i>GID</i></code>	This expression searches for files whose group ID (GID) is set to <i>GID</i> .
<code>--uid <i>UID</i></code>	This expression searches for files owned by the user whose user ID (UID) is <i>UID</i> .
<code>--maxdepth <i>levels</i></code>	If you want to search a directory and, perhaps, some limited number of subdirectories, you can use this expression to limit the search.

There are many variant and additional options; `find` is a very powerful command. As an example of its use, consider the task of finding all C source code files, which normally have names that end in `.c`, in all users’ home directories. If these home directories reside in `/home`, you might issue the following command:

```
# find /home -name "*.c"
```

The result will be a listing of all the files that match the search criteria.



Ordinary users may use `find`, but it doesn't overcome Linux's file permission features. If you lack permission to list a directory's contents, `find` will return that directory name and the error message `Permission denied`.

Using the *locate* Command

The `locate` utility works much like `find` if you want to find a file by name, but it differs in two important ways:

- The `locate` tool is far less sophisticated in its search options. You normally use it to search only on filenames, and the program returns all files that contain the specified string. For instance, when searching for `rpm`, `locate` will return other programs, like `gnorpm` and `rpm2cpio`.
- The `locate` program works from a database that it maintains. Most distributions automatically call `locate` with options that cause it to update its database periodically, such as once a night or once a week. (You can also use the `updatedb` command to do this task at any time.) For this reason, `locate` may not find recent files, or it may return the names of files that no longer exist. If the database update utilities omit certain directories, files in them won't be returned by a `locate` query.

Because `locate` works from a database, it's typically much faster than `find`, particularly on system-wide searches. It's likely to return many false alarms, though, especially if you want to find a file with a short name. To use it, type **`locate search-string`**, where *search-string* is the string that appears in the filename.



Some Linux distributions use `slocate` rather than `locate`. The `slocate` program includes security features to prevent users from seeing the names of files in directories they should not be able to access. On most systems that use `slocate`, the `locate` command is a link to `slocate`, so `locate` implements `slocate`'s security features. A few distributions don't install either `locate` or `slocate` by default.

Using the *whereis* Command

The `whereis` program searches for files in a restricted set of locations, such as standard binary file directories, library directories, and man page directories. This tool does *not* search user directories or many other locations that are easily searched by `find` or `locate`. The `whereis` utility is a quick way to find program executables and related files like documentation or configuration files.

The `whereis` program returns filenames that begin with whatever you type as a search criterion, even if those files contain extensions. This feature often turns up configuration

files in /etc, man pages, and similar files. To use the program, type the name of the program you want to locate. For instance, the following command locates `ls`:

```
$ whereis ls
ls: /bin/ls /usr/share/man/man1/ls.1.bz2
```

The result shows both the `ls` executable (`/bin/ls`) and the `ls` man page. The `whereis` program accepts several parameters that modify its behavior in various ways. These are detailed in the program’s man page.

Examining Files’ Contents

Locating files by name, owner, or other surface characteristics is very convenient, but sometimes you need to locate files based on their contents or quickly examine files without loading them into a text editor. Naturally, Linux provides tools to perform these tasks.

Finding Files by Content

The `grep` command is extremely useful. It searches for files that contain a specified string and returns the name of the file and (if it’s a text file) a line of context for that string. The basic `grep` syntax is as follows:

```
grep [options] pattern [files]
```

Like `find`, `grep` supports a large number of options. Table 2.7 summarizes some of the more common options.

TABLE 2.7 Common `grep` Options

Option Name	Option Abbreviation	Meaning
<code>--count</code>	<code>-c</code>	Instead of displaying context lines, <code>grep</code> displays the number of lines that match the specified pattern.
<code>--file=file</code>	<code>-f file</code>	This option takes pattern input from the specified file, rather than from the command line.
<code>--ignore-case</code>	<code>-i</code>	You can perform a case-insensitive search, rather than the default case-sensitive search, by using this option.
<code>--recursive</code>	<code>-r</code>	This option searches in the specified directory and all subdirectories, rather than simply the specified directory.

pattern is a *regular expression*, which can be a complex specification that can match many different strings. Alphabetic and numeric characters are interpreted in a literal way in a regular expression, but some others have special meaning. For instance, if you enclose a series of letters or numbers in square braces (`[]`), the system matches any one of those characters. Suppose you want to locate all the files in `/etc` that contain the strings `tty1` or `tty2`. You could enter the following command:

```
# grep tty[12] /etc/*
```

You can use `grep` in conjunction with commands that produce a lot of output in order to sift through that output for the material that's important to you. (Several examples throughout this book use this technique.) Suppose you want to find the process ID (PID) of a running `xterm`. You can use a pipe (described shortly in the section “Using Redirection and Pipes”) to send the result of a `ps` command (described in Chapter 3, “Managing Processes and Editing Files”) through `grep`; thus:

```
# ps ax | grep xterm
```

The result is a list of all running processes called `xterm`, along with their PIDs. You can even do this in series, using `grep` to further restrict the output on some other criterion, which can be useful if the initial pass still produces too much output.

Viewing Short Text Files and Combining Text Files

The `cat` program has nothing to do with feline pets. Rather, it's short for the word “concatenate,” and it's a tool for combining files, one after the other, and sending them to *standard output* (that is, your screen, GUI terminal, or remote login session; sometimes called *stdout*). One common use for `cat` is to forgo the multifile aspect of the command and display a single file. For instance, the following command displays the contents of `/etc/fstab`:

```
$ cat /etc/fstab
```

This can be a good way to quickly view a short file. It's much less effective for large files, though, because the start of the file will scroll off the top of the display. For very long files, it may also take a long time to display the entire file.

Another use of `cat` is to quickly combine two files into one. This is best achieved in conjunction with the redirection operator (`>`), which is described in more detail shortly. For instance, suppose you want to combine `/etc/fstab` with `/etc/fstab-addition`. You might issue the following command:

```
# cat /etc/fstab fstab-addition > fstab-plus
```

You could then examine the resulting file, `fstab-plus`. If `fstab-addition` contains a new entry you wanted to add to `/etc/fstab`, copying `fstab-plus` over the old `/etc/fstab` will accomplish the job. In fact, `cat` can even serve as a quick-and-dirty way to create a text file:

```
$ cat - > text.txt
```

The `-` character from which `cat` is reading is a shorthand for *standard input* (sometimes called *stdin*)—normally your keyboard. Anything you type after this point will be entered into `text.txt`, until you press `Ctrl+D`. This keystroke terminates the `cat` program, at which point `text.txt` will contain your desired text. This can be a particularly useful trick if you're using an extremely spare emergency system and need to quickly create a short configuration file.

Viewing Long Text Files

A program that's used in many OSs to enable users to view information in a controlled way is known as `more`. Typing `more filename` results in a screen-by-screen display of `filename`'s contents. You can press the Enter key to move down one line of text, or you can press the spacebar to move forward by one screen. When you're done, press the Q key to exit. This can be a convenient way to view configuration or other text files.

Although `more` is useful, the original program has many limitations. For instance, there's no way to page *backward* through a file or search for text within the file. These needs spawned a better version of `more`, which is known as `less` in a twist of humor. In addition to paging forward, `less` enables you to type in various keystrokes to do other things. Some of these are modeled after the keystrokes used in the Emacs editor, such as `Ctrl+V` to move forward by a screen and `Esc` followed by `V` to move backward by a screen. You can also search for text by typing `/` followed by the search pattern. Typing `q` exits from `less`. You can learn more from the `less` man page.



Most Linux systems use `less` to display man pages, so you can practice the `less` commands while viewing the `less` man page.

Viewing the Ends of Files

Sometimes you want to view the last few lines of a file but not the beginning of the file. For instance, you might want to check a log file to see whether an action you've just performed has created an entry. Because programs log actions at the ends of log files, a way to quickly check the end of the file is convenient. This was the purpose for which `tail` was written. It displays the last 10 lines of a file (or if you include the `-n num` parameter, the last `num` lines). For instance, to view the last 20 lines of `/var/log/messages`, you could type the following command:

```
# tail -n 20 /var/log/messages
```

You can also use `tail` to view a file as it's updated by another process. To do this, you use the `-f` or `--follow` option. This feature is useful for monitoring log files as you test the function of a program that you expect to write to the log file; you'll see its logged messages as soon as they appear in the log file.

Using Redirection and Pipes

Several of the preceding examples have used *redirection* and *pipes* (aka *pipelines*). These are mechanisms that you can use to redirect the input to a process or the output from a process. Redirection passes input to or from a file, and a pipe enables you to tie two or more programs together so that one uses the output of another as input.

Normally, the standard output of a program goes to the display you used to launch it. The output redirection operator, `>`, changes this, sending standard output to a file that you specify. For instance, suppose you want to capture the output of `ifconfig` in a file called `iface.txt`. You could use the following command to do this:

```
$ ifconfig > iface.txt
```

This operator wipes out the current `iface.txt` file, if it exists. If you want to append information rather than overwrite it, you can use the `>>` operator instead of `>`.

Many programs produce a second type of text-mode output, known as *standard error* (sometimes abbreviated *stderr*). This type of output carries error messages that should not be ignored, and so it's redirected separately from standard output. You use `2>` to redirect standard error, or `2>>` to append standard error to a file rather than replace that file. You can redirect both standard error and standard output by using `&>`.

You can replace standard input by using the input redirection operator, `<`. This is most useful when you must routinely provide the same information to a program time after time. You can create a file with that information and pass it to the program with the input redirection operator; thus:

```
$ superscript < script-input.txt
```

To have one program take another's output as input, you use a pipe, which is represented by a vertical bar (`|`). An earlier example illustrated this process: the output of `ps` may contain too much information to be quickly parsed, so you can pass its output through `grep` to locate just the information you want; thus:

```
# ps ax | grep xterm
```

This command searches for the string `xterm` in the `ps` output and displays all the lines that match. The output of `ps` goes into `grep`, and `grep`'s output appears on your screen. (You could use another pipe or redirect `grep`'s output, if you prefer.)

In addition to these basic redirection operators, you can use the `tee` command to create multiple copies of a program's output. It's typically used to send output both to standard output and to a file. To use it, pipe a program's output through `tee`, and give `tee` an output filename:

```
$ ps ax | tee processes.txt
```

This example displays the output of the `ps ax` command on the screen *and* stores it in the `processes.txt` file.

If you want to execute two commands in sequence but don't want to pipe them together, you can do so by separating the commands with a semicolon (;).

For instance, suppose you want to run `script1` followed immediately by `script2`. You could type the following command to do this:

```
$ script1 ; script2
```

Generating Command Lines

Sometimes you'll find yourself constructing a series of commands that are similar to each other but not similar enough to enable you to use their normal options to substitute a single command. For instance, suppose you want to remove every file in a directory tree with a name that ends in a tilde (~). (This filename convention denotes backup files created by certain text editors.) With a large directory tree, this task can be daunting; the usual file-deletion command (`rm`, described in more detail in Chapter 4, "Managing System Services") doesn't provide an option to search for and delete every file in a directory tree that matches such a specific criterion. One command that can do the search part of the job, though, is `find`, which is also described in more detail in Chapter 4. This command displays all the files that match criteria you provide. If you could combine the output of `find` to create a series of command lines using `rm`, the task would be solved. This is precisely the purpose of the `xargs` command.

The `xargs` command builds a command from its standard input. The basic syntax for this command is as follows:

```
xargs [options] [command [initial-arguments]]
```

command is the command you want to execute, and *initial-arguments* is a list of arguments you want to pass to the command. *options* are `xargs` options; they aren't passed to *command*. When you run `xargs`, it runs *command* once for every word passed to it on standard input, adding that word to the argument list for *command*. If you want to pass multiple options to the command, you can protect them by enclosing the group in quotation marks.

For instance, consider the task of deleting all those backup files, denoted by tilde characters. You can do this by piping the output of `find` to `xargs`, which then calls `rm`:

```
$ find ./ -name "*~" | xargs rm
```

The first part of this command (`find ./ -name "*~"`) finds all the files in the current directory (./) or its subdirectories with a name that ends in a tilde (*~). This list is then piped to `xargs`, which adds each one to its own `rm` command.

A tool that's similar to `xargs` in many ways is the backtick (`), which is a character to the left of the 1 key on most keyboards. The backtick is *not* the same as the single quote character ('), which is located to the right of the semicolon (;) on most keyboards.

Text within backticks is treated as a separate command whose results are substituted on the command line. For instance, to delete those backup files, you can type the following command:

```
$ rm `find ./ -name "*~" `
```

Using Device Files

A special class of files requires special attention: *Device files* are files that provide access to hardware. These files are typically located in `/dev` and its subdirectories. They include files to access hard disk devices (`/dev/sd*` and `/dev/hd*`), terminals (`/dev/tty*`), mice (`/dev/input/mouse*`), and so on.

A few device files are sometimes useful with redirection operators. The `/dev/null` file is not connected to anything, so data sent to `/dev/null` disappears forever. It's intended as a way to quickly discard unwanted output produced by a program. For instance, if the `whine` program is generating spurious error messages, you can redirect its standard error to `/dev/null`:

```
$ whine 2> /dev/null
```

The result is that you'll no longer see the spurious error messages—or any legitimate error messages!

The `/dev/zero` file generates an endless stream of 0 values. This is sometimes useful when you want to create a blank file. You'd typically use it with the `dd` utility, which copies a specified amount of data from one source to another. For instance, to create a 1MB blank file, you might type this:

```
$ dd if=/dev/zero of=empty.file bs=1024 count=1024
```

The `if` and `of` options to `dd` specify the input and output files, respectively. The `bs` and `count` options set the block size (the number of bytes read per operation) and the number of blocks, so setting both to 1024 copies 1024×1024 bytes, or 1MB.

The `/dev/random` and `/dev/urandom` files are another couple of special device files; they generate streams of random numbers, based on the random “noise” generated by physical hardware in the system. These files differ in that `/dev/random` will wait for more randomness to accumulate in the source hardware whenever it runs out, whereas `/dev/urandom` will not wait, which can produce an inferior random number stream. These files are of use to programs that need to generate random numbers (say, for cryptographic purposes).

Most Linux distributions use a dynamic device file directory tree, located at `/dev`. In most cases, you won't need to adjust these device files, since they're created automatically by a specialized device filesystem. On very old systems, though, or if you want to create device files outside of `/dev`, you can create device files with the `mknod` command, which has the following syntax:

```
mknod [options] name type [major minor]
```

The *name* you pass to `mknod` is the device filename, *type* is the hardware type (normally `b` for a block device, or `c` or `u` for a character device; but `p` is also valid, and creates a FIFO), and *major* and *minor* are the major and minor device numbers, respectively. A handful of *options* are valid and are described by the `mknod` man page.

Ordinarily, you'll only use `mknod` if you're an expert with device files. At the very least, you'll need to know the device type, major number, and minor number. This information is device-dependent, so you'll need to consult documentation on the device's kernel driver to learn what to use.

Using File Permissions

Linux uses a set of *file permissions*, or the file's *mode*, to determine how a file may be accessed. File permissions are an important part of Linux's user security, as described further in Chapter 5. To use these features, you must understand how Linux treats file permissions and what tools the OS provides for permission manipulation.

Understanding Accounts and Ownership

Chapter 5 describes Linux accounts in detail; however, file permissions interact with accounts, so before proceeding, you should understand the basics of Linux accounts. A Linux account is a set of data structures that programs and the kernel treat in a particular way in order to control access to the system. An account includes an alphanumeric username, a user ID (UID) number, a group ID (GID) number, information on the user's default shell, and so on. When you log into a Linux computer, you provide a username, and Linux thereafter associates all actions you perform with the account that matches the username you provided. File permissions enable you to specify what accounts may access your files, as well as what files and programs you may access as a given user.

In addition to accounts, Linux supports groups, which are collections of accounts. The system administrator defines a set of users who belong to a specific group. In addition to being owned by a particular user, each file is associated with a specific group, and permissions to the file may be defined for that group. This feature can be used to define different sets of users, such as people who are working together on a specific project, in order to give them access to project-related files while keeping other users from accessing those files.

Using File Access Permissions

File access permissions in Linux involve several components, which combine to determine who may access a file and in what way. These components also help you determine precisely what a file is—an ordinary data file, a program file, a subdirectory, or something else. You must understand this setup if you want to manipulate file permissions.

Understanding File Access Components

Linux's file permission handling has three components:

Username (or UID) A username (or UID, as it's stored in this form) is associated with each file on the computer. This is frequently referred to as the *file owner*.

Group (or GID) Every file is associated with a particular GID, which links the file to a group. This is sometimes referred to as the *group owner*. Normally, the group of a file is one of the groups to which the file's owner belongs, but *root* may change the file's group to one unassociated with the file's owner.

File access permissions The file access permissions (or *file permissions* or *mode* for short) are a code that represents who may access the file, relative to the file's owner, the file's group, and all other users.

You can see all three elements by using the `ls -l` command on a file:

```
$ ls -l /usr/sbin/lsof
-rwxr-xr-x 1 root kmem 84124 Oct 3 02:37 /usr/sbin/lsof
```

The output of this command has several components, each with a specific meaning:

Permission string The first component, `-rwxr-xr-x`, is the permission string. Along with the user and group names, it's what determines who may access a file. As displayed by `ls -l`, the permission string is a series of codes, which are described in more detail in the upcoming section "Interpreting File Access Codes." Sometimes the first character of this string is omitted, particularly when describing ordinary files, but it's always present in an `ls -l` listing.

Number of hard links Internally, Linux uses a data structure known as an *inode* to keep track of the file, and multiple filenames may point to the same inode, in the form of a hard link. The number 1 in the preceding example output means that just one filename points to this file; it has no hard links. Larger numbers indicate that hard links exist—for instance, 3 means that the file may be referred to by three different filenames.



Soft links are *not* referenced in the linked-to file's directory listing.

Owner The next field, *root* in this example, is the owner of the file. In the case of long usernames, the username may be truncated.

Group The example file belongs to the *kmem* group. Many system files belong to the *root* owner and *root* group; for this example, I picked a file that belongs to a different group.

File size The next field, 84124 in the preceding example, is the size of the file in bytes.

Creation time The next field contains the file creation date and time (Oct 3 02:37 in this example). If the file is older than a year, you'll see the year rather than the creation time, although the time is still stored with the file.

Filename The final field is the name of the file. Because the `ls` command in the preceding example specified a complete path to the file, the complete path appears in the output. If the command had been issued without that path but from the `/usr/sbin` directory, `ls` would appear alone.

Although information such as the number of hard links and file-creation date may be useful on occasion, it's not critical for determining file access rights. For this, you need the file's owner, group, and file access permission string.

Linux Filesystem Data Structures

Linux filesystems store several types of data. Most of the space is consumed by file data—the contents of word processing documents, spreadsheets, program executable files, and so on. In order to give you access to file data, though, the system also uses directories, which are lists of filenames. (In fact, directories are stored as ordinary files with special file attributes set.)

In order to link between a directory entry and a file, Linux filesystems use an inode. This is a special filesystem data structure that holds assorted data about the file, including a pointer to the location of the data on the disk, the file's mode, the UID, the GID, and so on. The directory entry points to the inode, which in turn points to the file data proper.

Filesystems also have free space bitmaps, which let the OS know which sectors on a disk have and have not been used. When storing a new file or expanding an existing one, Linux checks the free space bitmap to see where free space is available.

Not all filesystems use actual inodes. For instance, the File Allocation Table (FAT) filesystem used by DOS and Windows doesn't use inodes; instead, it places the information that's in Linux filesystems' inodes in the directory and in the free space bitmap. When Linux uses such a filesystem, it creates virtual inodes from the FAT data structures.

Interpreting File Access Codes

The file access control string is 10 characters in length. The first character has special meaning—it's the *file type code*. The type code determines how Linux will interpret the file—as ordinary data, a directory, or a special file type. Table 2.8 summarizes Linux type codes.

TABLE 2.8 Linux File Type Codes

Code	Meaning
-	Normal data file; may be text, an executable program, graphics, compressed data, or just about any other type of data.
d	Directory; disk directories are files just like any others, but they contain file-names and pointers to disk inodes.
l	Symbolic link; the file contains the name of another file or directory. When Linux accesses the symbolic link, it tries to read the linked-to file.
p	Named pipe; a pipe enables two running Linux programs to communicate with each other. One opens the pipe for reading, and the other opens it for writing, enabling data to be transferred between the programs.
s	Socket; a socket is similar to a named pipe, but it permits network and bidirectional links.
b	Block device; a file that corresponds to a hardware device to and from which data is transferred in blocks of more than one byte. Disk devices (hard disks, floppies, CD-ROMs, and so on) are common block devices.
c	Character device; a file that corresponds to a hardware device to and from which data is transferred in units of one byte. Examples include text-mode terminals, printers, and sound cards.

The remaining nine characters of the permission string (`rw-r-xr-x` in the example in the earlier section “Understanding File Access Components”) are broken up into three groups of three characters. The first group controls the file owner’s access to the file, the second controls the group’s access to the file, and the third controls all other users’ access to the file (often referred to as *world permissions*).

In each of these three cases, the permission string determines the presence or absence of each of three types of access: read, write, and execute. Read and write permissions are fairly self-explanatory, at least for ordinary files. If the execute permission is present, it means that the file may be run as a program. (Of course, this doesn’t turn a nonprogram file into a program; it means only that a user may run a program if it is a program. Setting the execute bit on a nonprogram file will probably cause no real harm, but it could be confusing.) The absence of the permission is denoted by a hyphen (-) in the permission string. The presence of the permission is indicated by a letter—*r* for read, *w* for write, or *x* for execute.

Thus, the example permission string of `rw-r-xr-x` means that the file’s owner, members of the file’s group, and all other users can read and execute the file. Only the file’s owner has write permission to the file. You can easily exclude those who don’t belong to the file’s group, or even all but the file’s owner, by changing the permission string, as described shortly in “Changing File Ownership and Permissions.”

Individual permissions, such as execute access for the file’s owner, are often referred to as *permission bits*. This is because Linux encodes this information in binary form. Because it is binary, the permission information can be expressed as a single 9-bit number. This number is usually expressed in octal (base 8) form because a base-8 number is 3 bits in length, which means that the base-8 representation of a permission string is 3 digits long, one digit for each of the owner, group, and world permissions. The read, write, and execute permissions each correspond to one of these bits. The result is that you can determine owner, group, or world permissions by adding base-8 numbers: 1 for execute permission, 2 for write permission, and 4 for read permission.

Table 2.9 shows some examples of common permissions and their meanings. This table is necessarily incomplete, though; with 9 permission bits, the total number of possible permissions is 2^9 , or 512. Most of those possibilities are peculiar, and you’re not likely to encounter or create them except by accident.

TABLE 2.9 Example Permissions and Their Meanings

Permission String	Octal Code	Meaning
<code>rw-rwxrwx</code>	777	Read, write, and execute permissions for all users.
<code>rw-r-xr-x</code>	755	Read and execute permission for all users. The file’s owner also has write permission.
<code>rw-r-x---</code>	750	Read and execute permission for the owner and group. The file’s owner also has write permission. Users who are not the file’s owner or members of the group have no access to the file.
<code>rw-----</code>	700	Read, write, and execute permissions for the file’s owner only; all others have no access.
<code>rw-rw-rw-</code>	666	Read and write permissions for all users. No execute permissions to anybody.
<code>rw-rw-r--</code>	664	Read and write permissions to the owner and group. Read-only permission to all others.
<code>rw-rw----</code>	660	Read and write permissions to the owner and group. No world permissions.
<code>rw-rw----</code>	660	Read and write permissions to the owner and group. No world permissions.
<code>rw-r--r--</code>	644	Read and write permissions to the owner. Read-only permission to all others.

TABLE 2.9 Example Permissions and Their Meanings (*continued*)

Permission String	Octal Code	Meaning
rw-r-----	640	Read and write permissions to the owner, and read-only permission to the group. No permission to others.
rw-----	600	Read and write permissions to the owner. No permission to anybody else.
r-----	400	Read permission to the owner. No permission to anybody else.

Execute permission makes sense for ordinary files, but it's meaningless for most other file types, such as device files. Directories, though, make use of the execute bit in another way. When a directory's execute bit is set, that means that the directory's contents may be searched. This is a highly desirable characteristic for directories, so you'll almost never find a directory on which the execute bit is *not* set in conjunction with the read bit.

Directories can be confusing with respect to write permission. Recall that directories are files that are interpreted in a special way. As such, if a user can write to a directory, that user can create, delete, or rename files in the directory, even if the user isn't the owner of those files and does not have permission to write to those files. The *sticky bit* (described shortly) can alter this behavior.

Symbolic links are unusual with respect to permissions. This file type always has 777 (rwxrwxrwx) permissions, thus granting all users full access to the file. This access applies only to the link file itself, however, not to the linked-to file. In other words, all users can read the contents of the link to discover the name of the file to which it points, but the permissions on the linked-to file determine its file access. Attempting to change the permissions on a symbolic link affects the linked-to file.

Many of the permission rules do not apply to *root*. The superuser can read or write any file on the computer—even files that grant access to nobody (that is, those that have 000 permissions). The superuser still needs an execute bit to be set to run a program file, but the superuser has the power to change the permissions on any file, so this limitation isn't very substantial. Some files may be inaccessible to *root* but only because of an underlying restriction—for instance, even *root* can't access a hard disk that's not installed in the computer.

A few special permission options are also supported, and they may be indicated by changes to the permission string:

Set user ID (SUID) The *set user ID (SUID)* option is used in conjunction with executable files, and it tells Linux to run the program with the permissions of whoever owns the file, rather than with the permissions of the user who runs the program. For instance, if a file is owned by *root* and has its SUID bit set, the program runs with *root* privileges and can

therefore read any file on the computer. Some servers and other system programs run in this way, which is often called SUID root. SUID programs are indicated by an *s* in the owner's execute bit position of the permission string, as in *rwsr-xr-x*. (As described in Chapter 3, SUID programs can pose a security risk.)

Set group ID (SGID) The *set group ID (SGID)* option is similar to the SUID option, but it sets the group of the running program to the group of the file. It's indicated by an *s* in the group execute bit position of the permission string, as in *rwxr-sr-x*.

Sticky bit The sticky bit has changed meaning during the course of Unix history. In modern Linux implementations (and most modern versions of Unix), it's used to protect files from being deleted by those who don't own the files. When this bit is present on a directory, the directory's files can be deleted only by their owners, the directory's owner, or root. The sticky bit is indicated by a *t* in the world execute bit position, as in *rwxr-xr-t*.

Determining File Types

Although the long form of *ls* displays some file type information, in the form of executable bits and the file type codes shown in Table 2.8, this information is limited. You can sometimes tell the type of a file from its filename extension—for instance, files with *.jpg* extensions are normally Joint Photographic Experts' Group (JPEG) graphics files, whereas *.txt* files are ordinarily American Standard Code for Information Interchange (ASCII; aka plain text) files. You must be familiar with the filename extensions to make this determination, though, and of course the filenames might be misnamed.

To help work around these problems, the *file* command examines a file and reports on the type of data it contains:

```
$ file disks.jpg
disks.jpg: ASCII text
$ file snowy-park.txt
snowy-park.txt: JPEG image data, JFIF standard 1.01
```

In this example, both these files are misnamed: the JPEG file has a *.txt* extension, whereas the ASCII file has a *.jpg* extension. The *file* command isn't fooled by this discrepancy. This command isn't infallible, though. It works from a database of file types, so any file type that's not in its database won't be correctly identified.

The *test* command is another tool for testing the status of a file. It can test for a file's existence, compare two files' creation times, and so on. This tool, however, is mainly used in shell scripts, rather than as an interactive tool.

Changing File Ownership and Permissions

Changing who can read, write, or execute a file can be done using several programs, depending on the desired effect. Specifically, *chown* changes a file's owner and, optionally, its group; *chgrp* changes the file's group; and *chmod* changes the permissions string.

Modifying Ownership

To begin, `chown`'s syntax is as follows:

```
chown [options] [newowner][:newgroup] filename...
```

The variables *newowner* and *newgroup* are, of course, the new owner and group of the file. One or both are required. If both are included, there must be no space between them, only a single colon (:). For instance, the following command gives ownership of the file `report.tex` to `sally`, and it sets the file's group to `project2`:

```
# chown sally:project2 report.tex
```



Old versions of `chown` used a period (.) instead of a colon. Current versions (through at least 6.1.0) still accept periods in this role, but they may complain about your use of an unfashionably old syntax.

The `chown` command supports a number of options, such as `--dereference` (which changes the referent of a symbolic link) and `--recursive` (which changes all the files within a directory and all its subdirectories). The latter is probably the most useful option for `chown`.

The `chgrp` command is similar to `chown`, except that it doesn't change or alter the file's owner—it works only on the group. The group name is *not* preceded by a period. For instance, to change the group of `report.tex` to `project2`, you could issue the following command:

```
# chgrp project2 report.tex
```

The `chgrp` command takes the same options as `chown` does. One caveat to the use of both commands is that even the owner of a file may not be able to change the ownership or group of a file. The owner may change the group of a file to any group to which the file's owner belongs, but not to other groups. Normally, only `root` may change the owner of a file.

Modifying Permissions

The `chmod` command changes a file's permissions. This command may be issued in many different ways to achieve the same effect. Its basic syntax is as follows:

```
chmod [options] [mode[,mode...]] filename...
```

The `chmod` options are similar to those of `chown` and `chgrp`. In particular, `--recursive` (or `-R`) will change all the files within a directory tree.

Most of the complexity of `chmod` comes in the specification of the file's mode. There are two basic forms in which you can specify the mode: as an octal number or as a symbolic mode, which is a set of codes related to the string representation of the permissions.

The octal representation of the mode is the same as that described earlier and summarized in Table 2.9. For instance, to change permissions on `report.tex` to `rw-r--r--`, you could issue the following command:

```
# chmod 644 report.tex
```

In addition, it's possible to precede the three digits for the owner, group, and world permissions with another digit that sets special permissions. Three bits are supported (hence values between 0 and 7): adding octal 4 sets the set user ID (SUID) bit, adding octal 2 sets the set group ID (SGID) bit, and adding octal 1 sets the sticky bit. If you omit the first digit (as in the preceding example), Linux clears all three bits. Using four digits causes the first to be interpreted as the special permissions code. For instance, suppose you've created a script called `bigprogram` in a text editor. You want to set both SUID and SGID bits (6); to make the script readable, writeable, and executable by the owner (7); to make it readable and executable by the group (5); and to make it completely inaccessible to all others (0). The following commands illustrate how to do this; note the difference in the mode string before and after executing the `chmod` command:

```
$ ls -l bigprogram
-rw-r--r-- 1 rodsmith users 10323 Oct 31 18:58 bigprogram
$ chmod 6750 bigprogram
$ ls -l bigprogram
-rwsr-s--- 1 rodsmith users 10323 Oct 31 18:58 bigprogram
```

A symbolic mode, by contrast, consists of three components: a code indicating the permission set you want to modify (the owner, the group, and so on); a symbol indicating whether you want to add, delete, or set the mode equal to the stated value; and a code specifying what the permission should be. Table 2.10 summarizes all these codes. Note that these codes are all case sensitive.

TABLE 2.10 Codes Used in Symbolic Modes

Permission Set Code	Meaning	Change Type Code	Meaning	Permission to Modify Code	Meaning
u	Owner	+	Add	r	Read
g	Group	-	Remove	w	Write
o	World	=	Set equal to	x	Execute
a	All			X	Execute only if file is directory or already has execute permission
				s	SUID or SGID
				t	Sticky bit
				u	Existing owner's permissions

TABLE 2.10 Codes Used in Symbolic Modes (*continued*)

Permission Set Code	Meaning	Change Type Code	Meaning	Permission to Modify Code	Meaning
				g	Existing group permissions
				o	Existing world permissions

To use symbolic permission settings, you combine one or more of the codes from the first column of Table 2.10 with one symbol from the third column and one or more codes from the fifth column. You can combine multiple settings by separating them by commas. Table 2.11 provides some examples of `chmod` using symbolic permission settings.

TABLE 2.11 Examples of Symbolic Permissions with `chmod`

Command	Initial Permissions	End Permissions
<code>chmod a+x bigprogram</code>	<code>rw-r--r--</code>	<code>rw-r-xr-x</code>
<code>chmod ug=rw report.tex</code>	<code>r-----</code>	<code>rw-rw----</code>
<code>chmod o-rwx bigprogram</code>	<code>rw-rwxr-x</code>	<code>rw-rwx---</code>
<code>chmod g=u report.tex</code>	<code>rw-r--r--</code>	<code>rw-rw-r--</code>
<code>chmod g-w,o-rw report.tex</code>	<code>rw-rw-rw-</code>	<code>rw-r-----</code>

As a general rule, symbolic permissions are most useful when you want to make a simple change (such as adding execute or write permissions to one or more class of users) or when you want to make similar changes to many files without affecting their other permissions (for instance, adding write permissions without affecting execute permissions). Octal permissions are most useful when you want to set some specific absolute permission, such as `rw-r--r--` (644). In any event, you should be familiar with both methods of setting permissions.

A file's owner and root are the only users who may adjust a file's permissions. Even if other users have write access to a directory in which a file resides and write access to the file itself, they may not change the file's permissions (but they may modify or even delete the file). To understand why this is so, you need to know that the file permissions are stored as part of the file's inode, which isn't part of the directory entry. Read/write access to the directory entry, or even the file itself, doesn't give a user the right to change the inode structures (except indirectly—for instance, if a write changes the file's size or a file deletion eliminates the need for the inode).

In addition to `chmod` and its file permissions, you can view and manipulate more fundamental permissions when using an `ext2`, `ext3`, or `ext4` filesystem.

The `lsattr` command displays file attributes, and `chattr` enables you to change them. Attributes adjustable in this way include automatic kernel compression (`c`), a flag to prevent backup by `dump` (`d`), zeroing of allocated data upon file deletion (`s`), and various other highly technical attributes. Consult the `lsattr` and `chattr` man pages for more information.

These two commands are useless on XFS, JFS, ReiserFS, or any other filesystem except for the `ext2` family.

Setting Default Permissions

When a user creates a file, that file has default ownership and permissions. The default owner is, understandably, the user who created the file. The default group is the user's primary group. The default permissions, however, are configurable. These are defined by the *user mask* (*umask*), which is set by the `umask` command. This command takes as input an octal value that represents the bits to be removed from 777 permissions for directories, or from 666 permissions for files, when creating a new file or directory. Table 2.12 summarizes the effect of several possible `umask` values.

TABLE 2.12 Sample Umask Values and Their Effects

Umask	Created Files	Created Directories
000	666 (rw-rw-rw-)	777 (rwxrwxrwx)
002	664 (rw-rw-r--)	775 (rwxrwxr-x)
022	644 (rw-r--r--)	755 (rwxr-xr-x)
027	640 (rw-r-----)	750 (rwxr-x---
077	600 (rw-----)	700 (rwx-----)
277	400 (r-----)	500 (r-x-----)

Note that the `umask` isn't a simple subtraction from the values of 777 or 666; it's a bit-wise removal. Any bit that's set in the `umask` is removed from the final permission for new files, but if the execute bit isn't set (as in ordinary files), its specification in the `umask` doesn't do any harm. For instance, consider the 7 values in several entries of Table 2.12's "Umask" column. This corresponds to a binary value of 111. An ordinary file might have `rw-` (110) permissions, but applying the `umask`'s 7 (111) eliminates 1 values but doesn't touch 0 values, thus producing a 000 (binary) value—that is, --- permissions, expressed symbolically.

Ordinary users can enter the `umask` command to change the permissions on new files they create. The superuser can also modify the default setting for all users by modifying a system configuration file. Typically, `/etc/profile` contains one or more `umask` commands. Setting the `umask` in `/etc/profile` might or might not actually have an effect, because it can be overridden at other points, such as a user's own configuration files. Nonetheless, setting the `umask` in `/etc/profile` or other system files can be a useful procedure if you want to change the default system policy. Most Linux distributions use a default `umask` of 002 or 022.

To find what the current `umask` is, type `umask` alone, without any parameters. Typing `umask -S` displays the `umask` expressed symbolically, rather than in octal form. You may also specify a `umask` in this way when you want to change it, but in this case, you specify the bits that you *do* want set. For instance, typing `umask u=rwx,g=rx,o=rx` is equivalent to typing `umask 022`.

Using ACLs

The Unix file permission system used by Linux was designed long ago. As frequently happens, real-world needs highlight limitations in early designs, and this is true of Unix permissions. For instance, using Unix permissions to provide fine-grained access control on a user-by-user basis is difficult or impossible. That is, if you want to enable the users `amy`, `david`, `theo`, and `loia` to read a file, but no other users, you must create a group that holds just those four users and no others, assign group ownership of the file to that group, and set group and world permissions appropriately. If you want only `amy` and `david` to be able to read another file, you must repeat this process, creating *another* group. What's more, because only `root` can ordinarily create groups, users have little control over these matters.

A more flexible approach is to use *access control lists (ACLs)*. These are permissions that can be assigned on a user-by-user basis. For instance, you can create ACLs allowing `amy` and `david` to access a file without creating or modifying any groups. In Linux, ACLs can provide separate read, write, and execute access; they are effectively an extension of the normal Unix-style permissions.

One problem with ACLs is that they're not yet universally supported. The filesystem you use must support ACLs. In modern Linux kernels, the `ext2fs`, `ext3fs`, `ext4fs`, `ReiserFS`, `JFS`, and `XFS` filesystems all support ACLs. This support may not be present in more obscure filesystems or in older kernels, though. ACL support is optional with all of the filesystems that support it; it must be enabled when the kernel is compiled, and this isn't always done.

Assuming your Linux filesystem supports ACLs, both `root` and ordinary users may create ACLs using the `setfacl` command:

```
setfacl [options] [{-m | -x} acl_spec] [{-M | -X} acl_file] file
```

The `-m` and `-M` parameters set an ACL, while the `-x` and `-X` parameters remove an ACL. The uppercase variants take an ACL out of a file, whereas the lowercase variants require you to enter the ACL on the command line. The ACL format itself takes this form:

```
scope:ID:perms
```

In this case, *scope* is the type of entity to which the ACL applies—typically *u* or *user* for a user, but *g* or *group* for a group is also possible, as is *o* or *others* to set a world ACL. The *ID* is a UID, GID, username, or group name. (This component is omitted if you use a *scope* of *other*.) The *perms* field specifies the permissions, either in octal form (as a single digit from 0 to 7) or in symbolic form (for instance, *rw-* or *r-x*).

As an example, consider a user who wants to create an ACL to enable another user (theo) to be able to read a file:

```
$ setfacl -m user:theo:r-- dinosaur.txt
```

Once this command is issued, *theo* can read the *dinosaur.txt* file, even if the file's ordinary Unix permissions would not permit this access. Because ordinary users may create and modify ACLs on their own files, the system administrator need not be bothered to create new groups. Users can create and modify ACLs only on files they own, though, much as they can modify the Unix permissions only on their own files.

If you want to see the ACLs on a file, you can use the *getfacl* command:

```
$ getfacl dinosaur.txt
# file: dinosaur.txt
# owner: amy
# group: users
user::rw-
user:theo:rw-
group:---
mask::rw-
other:---
```

Many Linux systems still have no need for ACLs; Unix-style permissions are adequate for many purposes. As more Linux tools are written to take full advantage of ACLs, they're likely to become more important.

Setting Environment Variables

Computer programs, like people, exist in certain environments. The environments for computer programs are defined by features such as the current working directory, the user's preferred language, and the type of terminal in use. In order to deliver this information to programs, Linux maintains *environment variables*. Like street signs in the physical world, environment variables convey information about the virtual "location" of the program and the resources that are available to it. Therefore, understanding how to set and use environment variables is important for both system administrators and users.

Programs query environment variables to learn about the state of the computer as a whole or what resources are available. These variables contain information such as the

location of the user's home directory, the computer's Internet hostname, and the name of the command shell that's in use. Individual programs may also use program-specific environment variables to tell them where their configuration files are located, how to display information, or how to use other program-specific options. As a general rule, though, environment variables provide information that's useful to multiple programs. Program-specific information is more often found in program configuration files.

Where to Set Environment Variables

If you're using Bash, you can set an environment variable from a command prompt for a specific login by typing the variable name followed by an equal sign (=) and the variable's value and then typing **export** and the variable name on the next line. For instance, you could type the following:

```
$ NNTPSERVER=news.abigisp.com
$ export NNTPSERVER
```

The first line sets the environment variable in your shell, and the second makes it available to programs you launch from the shell. You can shorten this syntax to a single line by typing **export** at the start of the first line:

```
$ export NNTPSERVER=news.abigisp.com
```

The former syntax is sometimes preferable when setting multiple environment variables because you can type each variable on a line and then use a single **export** command to make them all available. This can make shorter line lengths than you would get if you tried to export multiple variables along with their values on a single line. For instance, you could type the following:

```
$ NNTPSERVER=news.abigisp.com
$ YACLPATH=/usr/src/yac1
$ export NNTPSERVER,YACLPATH
```

This syntax is the same as that used for setting environment variables in `/etc/profile`. This system-wide configuration file is a Bash shell script, which means it contains commands that could be typed at a command prompt.



NOTE

When setting environment variables in a shell script such as `/etc/profile`, you should ignore the command prompts (\$) shown in these examples.

The preceding examples assigned values to environment variables. In other contexts, though, the environment variable is preceded by a dollar sign (\$). You can use this notation

to refer to an environment variable when setting another. For instance, in Bash, the following command adds `/opt/bin` to the existing `PATH` environment variable:

```
$ export PATH=$PATH:/opt/bin
```

In addition to the system-wide files, individual users may set environment variables by editing their local configuration files, such as `.bashrc` or `.bash_profile` for Bash.

The Meanings of Common Environment Variables

You may encounter many common environment variables on your system. You can find out how environment variables are configured by typing `env`. This command is used to run a program with a changed set of environment variables, but when it is typed alone, it returns all the environment variables that are currently set, in a format similar to that of Bash environment variable assignments:

```
NNTPSERVER=news.abigisp.com
```

Of course, the variables you see and their values will be unique to your system and even your account—that's the whole point of environment variables. Table 2.13 summarizes important variables you may see in this output.



Real World Scenario

Environment Variables in `tcsh`

Not all Linux users like Bash; some prefer `tcsh`, and a few use still other shells. The syntax used to set environment variables in Bash doesn't work for `tcsh`. For this shell, the appropriate command to set an environment variable is `setenv`. It's used much like `export` in its single-line form but without an equal sign:

```
$ setenv NNTPSERVER news.abigisp.com
```

When modifying an existing variable, you must add quotes around the new value and use curly braces around the original variable reference:

```
$ setenv PATH "${PATH}:/opt/bin"
```

Instead of using `/etc/profile`, `tcsh` uses the `/etc/csh.cshrc` and `/etc/csh.login` files for its system-wide configuration. Therefore, if your system has both Bash and `tcsh` users, you'll need to modify both files, using the appropriate syntax for each file. The user configuration files for `tcsh` are `.tcshrc`, `.cshrc`, or `.login` (`tcsh` tries each of these files in turn until it finds one that exists.)

TABLE 2.13 Common Environment Variables and Their Meanings

Variable Name	Explanation
USER	This is your current username. It's a variable that's maintained by the system.
SHELL	This variable holds the path to the current command shell.
PWD	This is the present working directory. This environment variable is maintained by the system. Programs may use it to search for files when you don't provide a complete pathname.
HOSTNAME	This is the current TCP/IP hostname of the computer.
PATH	This is an unusually important environment variable. It sets the <i>path</i> , which is a colon-delimited list of directories in which Linux searches for executable programs when you type a program name. For instance, if PATH is /bin:/usr/bin and you type <code>ls</code> , Linux looks for an executable program called <code>ls</code> in /bin and then in /usr/bin. If the command you type isn't on the path, Linux responds with a command not found error. The PATH variable is typically built up in several configuration files, such as /etc/profile and the .bashrc file in the user's home directory.
HOME	This variable points to your home directory. Some programs use it to help them look for configuration files or as a default location in which to store files.
LD_LIBRARY_PATH	A few programs use this environment variable to indicate directories in which library files may be found. It works much like PATH.
PS1	This is the default prompt in Bash. It generally includes variables of its own, such as \u (for the username), \h (for the hostname), and \W (for the current working directory). This value is frequently set in /etc/profile, but it is often overridden by users.
PS2	This is the prompt for a command completion. That is, if you press the Enter key and the shell recognizes that the command is incomplete (say, because you included an open quote but not the matching close quote), the shell will present the value of this environment variable as a prompt for you to complete typing the command.
PAGER	Programs that use a pager typically use the pager referred to by this environment variable. It usually refers to less, but you can redefine it if you like.
NNTPSERVER	Some Usenet news reader programs use this environment variable to specify the name of the news server system. This value might be set in /etc/profile or in the user's configuration files.

TABLE 2.13 Common Environment Variables and Their Meanings (*continued*)

Variable Name	Explanation
TERM	To handle more than basic text-mode effects, Linux has to know what commands the terminal supports. The TERM environment variable specifies the terminal in use, which is combined with information from additional files to provide terminal-specific code information. TERM is normally set automatically at login, but in some cases you may need to change it.
DISPLAY	This variable identifies the display used by X. It's usually :0.0, which means the first (numbered from 0) display on the current computer. When you use X in a networked environment, though, this value may be preceded by the name of the computer at which you're sitting, as in machine4.example.com:0.0. This value is set automatically when you log in, but you may change it if necessary. You can run multiple X sessions on one computer, in which case each one gets a unique DISPLAY number—for instance, :0.0 for the first session and :1.0 for the second.
EDITOR	Some programs launch the program pointed to by this environment variable when they need to call a text editor for you to use. Thus, changing this variable to your favorite editor can help you work in Linux. It's best to set this variable to a text-mode editor; GUI editors might cause problems if they're called from a program that was launched from a text-mode login.
PRINTER	You can set the default printer using this environment variable. (You can also set the default printer for the computer as a whole using the printing system, but this variable overrides the system-wide default when individual users set it.)



The PATH variable sometimes includes the current directory indicator (.) so that programs in the current directory can be run. This practice poses a security risk, though, because a miscreant could create a program with the name of some other program (such as ls) and trick another user into running it by leaving it in a directory the victim frequents. Even the root user may be victimized in this way. For this reason, it's best to omit the current directory from the PATH variable, especially for the superuser. If it's really needed for ordinary users, put it at the *end* of the path.

Any given system is likely to have several other environment variables set, but these are fairly esoteric or relate to specific programs. If a program's documentation says that it needs certain environment variables set, you can set them system-wide in /etc/profile or some other suitable file, or you can set them in user configuration files.

Although you can see the entire environment by typing **env**, this output can be long enough to be intimidating. If you just want to know the value of one variable, you can use the **echo** command, which echoes what you type to the screen. If you pass it a variable name preceded by a dollar sign (\$), **echo** returns the value of the variable. For instance:

```
$ echo $PS1
[\u@\h \W]\$
```

This command reveals that the **PS1** environment variable is set to `[\u@\h \W]\$`, which in turn produces a Bash prompt like `[david@penguin homes]$`.

Bash is a very advanced tool with some very esoteric features. In fact, Bash includes programming features (a topic touched upon briefly in the next section). One of these features is the equality operator, `==`, which you can use to test the equality of two variables, or a variable and a constant. For instance, suppose you want to execute a command only under certain conditions. You could do so like this:

```
$ if [[ $YOURVAR == "something" ]] ; then ls ; fi
```

This example executes the `ls` command if `$YOURVAR` is equal to `something`.

Advanced Bash users may employ conditional expressions in this way on a single command line, but this feature is more often used in shell scripts.

Using Shell Scripts

Bash, like most Linux shells, enables you to create *shell scripts*. A script is a program that's run directly from its source code (that is, the file created by the programmer) by an interpreter. This differs from a compiled program, which is converted from source code into a *binary* or *executable* form, which can be interpreted by the computer's CPU. Scripts tend to be quick to program because there's no need to compile them; but because the interpreter program must translate between source code and an executable form on the fly, scripts run more slowly than do compiled programs.

Shell scripts (scripts run by a shell) are popular for automating simple administrative tasks. For instance, you might write a shell script to create a large number of new accounts based on usernames in a text file or to configure the network in a particular way. Many of Linux's startup procedures are handled by shell scripts. On most distributions, `/etc/rc.d`, `/etc/init.d`, or subdirectories of these directories contain startup scripts that launch important system components, such as the printing subsystem and the GUI login prompt. (These startup scripts are covered in more detail in Chapter 4.)

Most Linux systems use the Bash shell by default, so shell scripts are often written in the Bash shell scripting language, but the `tcsh` and other shell scripting languages are quite similar. In fact, it's not uncommon to see shell scripts that run in any common Linux shell. You're not restricted to running shell scripts written in your default shell; the first line of a shell script identifies the shell that should be used to run it.



Like any programming task, shell scripting can be quite complex. The Linux+ 2009 exam, and therefore this chapter, covers only enough of this topic for you to write a very rudimentary shell script. Consult a book on the topic, such as *Mastering Unix Shell Scripting: Bash, Bourne, and Korn Shell Scripting for Programmers, System Administrators, and UNIX Gurus* by Randal K. Michael (Wiley, 2008), for more information.

Shell scripts are plain-text files, so you create them in text editors. A shell script begins with a line that identifies the shell that's used to run it, such as the following:

```
#!/bin/sh
```

The first two characters are a special code that tells the Linux kernel that this is a script and to use the rest of the line as a pathname to the program that's to interpret the script. Shell scripting languages use a hash mark (#) as a comment character, so the script utility itself ignores this line, although the kernel doesn't. On most systems, `/bin/sh` is a symbolic link that points to `/bin/bash`, but it could point to some other shell. Specifying the script as using `/bin/sh` guarantees that any Linux system will have a shell program to run the script, but if the script uses any features specific to a particular shell, you should specify that shell instead—for instance, use `/bin/bash` or `/bin/tcsh` instead of `/bin/sh`.

When you're done writing the shell script, you should modify it so that it's executable. You do this with the `chmod` command, as described earlier in “Modifying Permissions.” Specifically, you use the `+x` option to add execute permissions, probably in conjunction with a `a` to add these permissions for all users. For instance, to make a file called `my-script` executable, you'd issue the following command:

```
$ chmod a+x my-script
```

You'll then be able to execute the script by typing its name, possibly preceded by `./` to tell Linux to search in the current directory for the script. If you fail to make the script executable, you can still run the script by running the shell program followed by the script name (as in **`bash my-script`** or **`sh my-script`**), but it's generally better to make the script executable. If the script is one you run regularly, you may want to move it to a location on your path, such as `/usr/local/bin`. When you do that, you won't have to type the complete path or move to the script's directory to execute it; you can just type the script's name.

Shell scripts are composed of commands that the shell would recognize if typed at a command prompt. These can include both internal and external commands. You can even call other shell scripts from your own. For instance, suppose you want to start a script that launches two `xterms` and the KMail mail reader program. Listing 2.1 presents a shell script that accomplishes this goal.

Listing 2.1: A Simple Script That Launches Three Programs

```
#!/bin/bash
/usr/bin/xterm &
```

```
/usr/bin/xterm &
/usr/bin/kmail &
```

Aside from the first line that identifies it as a script, the script looks just like the commands you might type to accomplish the task manually, except for one fact: the script lists the complete paths to each program. This is usually not strictly necessary, but listing the complete path ensures that the script will find the programs even if the `PATH` environment variable changes. Also, each program-launch line in Listing 2.1 ends in an ampersand (&). This character tells the shell to run the program in the background and go on to the next line without waiting for the first program to terminate. If you omit the ampersands in Listing 2.1, the effect will be that the first `xterm` will open, but the second won't open until the first is closed. Likewise, `KMail` won't start until the second `xterm` is stopped.

Shell scripts can be much more complex than Listing 2.1—they can use all the usual programming tools, such as variables, loops, and conditional expressions. If you want to write such complex scripts (as, sooner or later, you will if you administer a Linux system professionally), you should consult additional documentation on this topic.

Getting Help

Nobody can know everything there is to know about Linux—the number of programs, each with its own set of options and features, is simply too great. For this reason, documentation and help resources come with Linux and are available online. One of the oldest forms of help is the manual page system, referred to as *man pages* for short. A somewhat newer tool for accessing similar documentation is known as *info pages*. Both of these systems are designed to provide you with quick summary information about a program. Neither system is intended to provide comprehensive tutorial information; for that, you must typically turn to other documentation that ships with programs or to third-party documentation. Some of these resources are available on the Internet, so knowing where to look for such help is critical.

Using Man Pages

Man pages provide succinct summaries of program functions. In the simplest case, they can be accessed by typing `man` followed by the name of a command, configuration file, system call, or other keyword. Each man page falls into one of nine categories, as summarized in Table 2.14. Some keywords lead to entries in multiple sections. In such instances, the `man` utility returns the entry for the lowest-numbered matching section by default. You can override this behavior by passing a section number before the keyword. For instance, typing `man passwd` returns information from manual section 1, on the `passwd` command, but typing `man 5 passwd` returns information from manual section 5, on the `/etc/passwd` file format. Some man pages have entries in sections with variant numbers that include the suffix `p`, as in section `1p`. These refer to POSIX standard man pages, as opposed to the Linux man pages, which are, for the most part, written by the people who wrote the open source Linux programs the man pages describe.

TABLE 2.14 Manual Sections

Section Number	Description
1	Executable programs and shell commands
2	System calls provided by the kernel
3	Library calls provided by program libraries
4	Device files (usually stored in /dev)
5	File formats
6	Games
7	Miscellaneous (macro packages, conventions, and so on)
8	System administration commands (programs run mostly or exclusively by root)
9	Kernel routines

The convention for man pages is a succinct style that employs several sections. Common sections include the following:

Name A man page begins with a statement of the command, call, or file that’s described, along with a few words of explanation. For instance, the man page for `man` (section 1) has a “Name” section that reads `man - an interface to the on-line reference manuals`.

Synopsis The synopsis provides a brief description of how a command is used. This synopsis uses a summary format similar to that used to present synopses in this book, showing optional parameters in square brackets (`[]`), for instance.

Description The description is an English-language summary of what the command, file, or other element does. The description can vary from a very short summary to something many pages in length.

Options This section summarizes the options outlined in the “Synopsis” section. Typically, each option appears in a list, with a one-paragraph explanation indented just below it.

Files This section lists files that are associated with the man page’s subject. These might be configuration files for a server or other program, related configuration files for a configuration file, or what have you.

See also This section provides pointers to related information in the man system, typically with a section number appended. For instance, `less(1)` refers to the section 1 man page for `less`.

Bugs Many man pages provide a “Bugs” section in which the author describes any known bugs or states that no known bugs exist.

History Some man pages provide a summary of the program’s history, citing project start dates and major milestones between then and the current version. This history isn’t nearly as comprehensive as the changes file that ships with most programs’ source code.

Author Most man pages end with an “Author” section, which tells you how to contact the author of the program.

Specific manual pages may contain fewer, more, or different sections than these. For instance, the “Synopsis” section is typically omitted from man pages on configuration files. Man pages with particularly verbose descriptions often split the “Description” section into several parts, each with its own title.

Man pages can be an extremely helpful resource, but you must understand their purpose and limitations. Unlike the help systems in some OSs, Linux man pages are not supposed to be either comprehensive or tutorial in nature; they’re intended as quick references to help somebody who’s already at least somewhat familiar with the subject. They’re most useful when you need to know the options to use with a command, the format of a configuration file, or similar summary information. If you need to learn a new program from scratch, other documentation is often a better choice. Man pages also vary greatly in quality; some are very good, but others are frustratingly terse and even occasionally inaccurate. For the most part, they’re written by the programmers who wrote the software in question, and programmers seldom place a high priority on user documentation.

Linux’s man pages use the `less` pager to display information. This pager’s operation is covered earlier in this chapter, in “Viewing Long Text Files.” Of course, you can also consult the `less` man page by typing `man less`. The upshot of using `less` is that you can page forward and backward, perform searches, and use other `less` functions when reading man pages. When you’re done using the man page system, press the Q key. This breaks you out of the `less` browser and returns you to your shell prompt.



Although `man` is a text-mode command, GUI variants exist. The `xman` program, for instance, provides a point-and-click method of browsing through man pages. You can’t type a subject on the command line to view it as you would with `man`, though—you must launch `xman` and then browse through the manual sections to a specific subject.

One of the problems with man pages is that it can be hard to locate help on a topic unless you know the name of the command, system call, or file you want to use. Fortunately, methods of searching the manual database exist and can help lead you to an appropriate man page:

Summary search The `whatis` command searches summary information contained in man pages for the keyword you specify. The command returns the “Name” section of every matching man page. You can then use this information to locate and read the man page you need. This command is most useful for locating all the man pages on a topic. For instance,

typing **what is man** returns lines confirming the existence of the man page entries for man, in sections 1, 5, 7, and 1p on a Fedora 10 system. (The entries available may vary from one distribution or installation to another.) The **what is** command's database is updated by the **makewhatis** command. This command is normally run automatically on a regular basis, but you can run it manually if the need arises.

Thorough search The **apropos** command performs a more thorough search, of both the “Name” and “Description” sections of man pages. The result looks much like the results of a **what is** search, except that it's likely to contain many more results. In fact, doing an **apropos** search on a very common word, such as **the**, is likely to return so many hits as to make the search useless. A search on a less common word is likely to be more useful. For instance, typing **apropos samba** returns just over a dozen entries on a Fedora 10 system, including those for **cupsaddsmb**, **smbpasswd**, and **smbstatus**—all tools related to the Samba file- and printer-sharing tool. (The exact number of hits returned by **apropos** will vary from system to system, depending on the packages installed.)

Using Info Pages

Linux's info page system is conceptually similar to its man page system, and info pages tend to be written in a similar terse style. The primary difference is that the info system uses a more sophisticated tool for presenting the documentation. Rather than a simple **less** browser on a linear file, the **info** command uses a more sophisticated hyperlinked format, conceptually similar to Web pages. The standard info browser, though, runs in text mode, so instead of clicking help items with your mouse, you must select them with the cursor keys or move about using keyboard shortcuts.



Some tools for reading info pages support mouse operations. The Emacs editor, for instance, includes a mouse-aware info reading tool. The **tkinfo** program (<http://math-www.uni-paderborn.de/~axel/tkinfo/>) is a general-purpose X-based info browser.

Info pages are written in *nodes*, which are similar to the individual pages of Web sites. These nodes are arranged hierarchically. To move from one node to another in the standard text-based info browser, you use any of several commands or procedures:

Next page Press the N key to move to the next node in a linked series of nodes on a single hierarchical level. This action may be required if the author intended several nodes to be read in a particular sequence.

Previous page Pressing the P key moves you back in a series of nodes on a single hierarchical level. This can be handy if you've moved forward in such a series but find you need to review earlier material.

Moving up Pressing the U key moves you up in the node hierarchy.

Selecting a topic To move down in the list of nodes, you select a topic and move into it. In the text-mode info browser, topics have asterisks (*) to the left of their names. You use your cursor keys to highlight the topic and press the Enter key to read that topic.

Last topic Pressing the L key displays the last info page you read. This action can move you up, down, or sideways in the info tree hierarchy.

Top page You can return to the top page for a topic (typically the one on which you entered the system) by pressing the T key.

Exiting When you're done using the info system, press the Q key.

On the whole, info pages can be more difficult to navigate than man pages, at least for the uninitiated; however, the hierarchical organization of information in info pages can make them superior tools for presenting information—there's less need to scroll through many pages of potentially uninteresting information looking for some tidbit. If the info page hierarchy was constructed sensibly, you should be able to find the information you need very efficiently.

Broadly speaking, programs sponsored by the Free Software Foundation (FSF) use info pages in preference to man pages. Many FSF programs now ship with minimal man pages that point the user to the programs' info pages. Non-FSF programmers have been slower to embrace info pages, though; many such programs don't ship with info pages at all and instead rely on traditional man pages. The info browser, though, can read and display man pages, so using info exclusively can be an effective strategy for reading Linux's standard documentation.

Using Miscellaneous Program Documentation

Most Linux programs ship with their own documentation, even aside from man or info pages. In fact, some programs have so much documentation that it's installed as a separate package, typically with the word `documentation` or `doc` in the package name, such as `samba-doc`.

The most basic and traditional form of program documentation is a file called `README`, `readme.txt`, or something similar. Precisely what information this file contains varies greatly from one program to another. For some, the file is so terse it's nearly useless. For others, it's a treasure trove of help. These files are almost always plain-text files, so you can read them with `less` or your favorite text editor.

If you downloaded the program as a source code tarball from the package maintainer's site, the `README` file typically appears in the main build directory extracted from the tarball. If you installed the program from a binary package file, though, the `README` file could be in any of several locations. The most likely places are `/usr/doc/packageName`, `/usr/share/doc/packageName`, and `/usr/share/doc/packages/packageName`, where *packageName* is the name of the package (sometimes including a version number, but more often not). If you can't find a `README` or similar file, use your distribution's package management system to locate documentation. For instance, on an RPM-based system, you might type `rpm -ql apackage | grep doc` to locate documentation for *apackage*. Using `grep` to search for the string `doc` in the file list is a good trick because documentation directories almost always contain the string `doc`. Chapter 7 describes `rpm` and other package-management commands in more detail.

**NOTE**

README files often contain information on building the package or make assumptions about binary file locations that don't apply to binaries provided with a distribution. Distribution maintainers seldom change such information in their README files, though.

In addition to or instead of the README file, many programs provide other documentation files. These may include a file that documents the history of the program in fine detail, descriptions of compilation and installation procedures, information on configuration file formats, and so on. Check the source code's build directory or the directory in which you found the README file for other files.

Some larger programs ship with extensive documentation in PostScript, Portable Document Format (PDF), Hypertext Markup Language (HTML), or other formats. Depending on the format and package, you might find a single file or a large collection of files. As with the README files, these files are well worth consulting, particularly if you want to learn to use a package to its fullest.

Using Internet-Based Help Resources

In addition to the documentation you find on your computer, you can locate documentation on the Internet. Most packages have associated Internet Web sites, which may be referred to in man pages, info pages, README files, or other documentation. Frequently, online documentation ships with the software, so you might be able to find it on your local hard disk; however, sometimes the local documentation is old or sparse compared to what's available online. Of course, if your local documentation is old, your local software may be old, too.

Another online resource that's extremely helpful is the Linux Documentation Project (LDP; <http://tldp.org>). The LDP is dedicated to providing more tutorial information than is commonly available with most Linux programs. You'll find several types of information at this site:

HOWTOs Linux HOWTO documents are short and medium-length tutorial pieces intended to get you up to speed with a topic or technology. HOWTOs have varying focus—some describe particular programs, whereas others are more task-oriented and cover a variety of tools in service to the task. As the name implies, they're generally designed to tell you how to accomplish some goal.

Guides Guides are longer documents, often described as book-length. (In fact, some of them are available in printed form.) Guides are intended as thorough tutorials or reference works on large programs or general technologies, such as Linux networking as a whole.

FAQs A *Frequently Asked Question (FAQ)* is, as the name implies, a question that comes up often—or more precisely, in the sense of the LDP category, that question and an answer to it. LDP FAQs are organized into categories, such as the Ftape FAQ or the WordPerfect on Linux FAQ. Each contains multiple questions and their answers, often grouped in subcategories. If you have a specific question about a program or technology, looking for an appropriate FAQ can be a good place to look first for an answer.

LDP documents vary greatly in their thoroughness and quality. Some (particularly some of the guides) are incomplete; you can click a section heading and see an empty page or a comment that the text has yet to be written. Some LDP documents are very recent, but others are outdated, so be sure to check the date of any document before you begin reading—if you don't, you might end up doing something the hard way, or in a way that no longer works. (Most of the FAQs seem to be languishing; many have last-revision dates in the 1990s!) Despite these flaws, the LDP can be an excellent resource for learning about specific programs or about Linux generally. The better LDP documents are excellent, and even those of marginal quality often present information that's not obvious from man pages, info pages, or official program documentation.



Most Linux distributions include the LDP documents in one or more special documentation packages. Check your `/usr/doc` and `/usr/share/doc` directories for these files. If they're not present, look for likely packages on your installation media. If you have fast always-up Internet access, though, you might want to use the online versions of LDP documents because you can be sure they're the latest available. Those that ship with a distribution can be weeks or months out of date by the time you read them.

Summary

Linux has strong historical ties to text-mode commands, and in fact Linux systems can be administered entirely from a text-mode login. Furthermore, even GUI tools in Linux are often front-ends to text-mode commands. For these reasons, familiarity with text-mode Linux tools is important for any Linux system administrator, and even for some users.

Text-mode use begins with an understanding of text-mode shells, such as Bash and `tcsh`. Shells accept text-mode commands and display their results, so knowing how to use a shell is necessary for effective use of a Linux system.

Once you've mastered shell basics, you can move on to basic file manipulation commands. These commands support navigating through Linux directories, moving and copying files, manipulating directories, locating files, and examining files. Using redirection and pipes with such commands is also a useful skill to possess.

Environment variables represent another key in text-mode Linux use. They can be set on a system-wide basis to control certain aspects of a user's Linux experience, such as the default prompt. Users can adjust their environment variables by typing appropriate commands or by editing their personal startup files.

Many system administration tasks involve repetitive actions. For this reason, most administrators learn to write at least basic shell scripts, which can combine many commands in one, frequently using variables and conditional expressions to improve the flexibility of the scripts.

Exam Essentials

Summarize how redirection operators and pipes can be useful. Redirection operators send a program's output to a file or send a file's contents to a program as input, enabling you to save a diagnostic tool's output for later perusal or give consistent input to a program. Pipes enable you to link together multiple programs, giving you more flexible and powerful multicommand tools.

Describe how files are moved and renamed in Linux. The `mv` command performs both of these tasks. When used on a single low-level filesystem, it changes disk pointers so that a file's location or name is changed, without altering or copying the file's data. When used across low-level filesystems, `mv` must copy the data, though.

Explain how directories are created and deleted in Linux. The `mkdir` command creates directories. Empty directories can be deleted with `rmdir`, or directory trees (including any files they contain) can be deleted with `rm`, by passing it the `-r` parameter.

Describe the differences between hard and symbolic links. Hard links are multiple directory entries that point to a single file. Symbolic links are special files that point to other files by filename.

Summarize the Linux ownership and permissions system. Files are owned by an individual account and are also associated with one group. Permission bits enable the file's owner to control separately the read, write, and execute access for the file's owner, members of the file's group, and all other users.

Describe when you might use `find` versus `grep`. The `find` command locates files based on surface features—the filename, file creation date, owner, and so on. The `grep` command reads the file's contents and enables you to search for files based on those contents.

Summarize the purpose of environment variables. Environment variables provide information that should be invariant across programs, such as the user's name and the path to be searched for program files.

Describe how a shell script can be useful. A shell script combines several commands, possibly including conditional expressions, variables, and other programming features to make the script respond dynamically to a system. Therefore, a shell script can reduce administrative effort by performing a series of repetitive tasks at one command.

Review Questions

1. Which of the following will add `/usr/local/bigprog/bin` to the end of the `PATH` environment variable, if placed in `/etc/profile`?
 - A. `export PATH=/usr/local/bigprog/bin`
 - B. `setenv PATH=$PATH:/usr/local/bigprog/bin`
 - C. `export PATH=$PATH:/usr/local/bigprog/bin`
 - D. `setenv PATH "${PATH}:/usr/local/bigprog/bin"`
2. Who may set default environment variables for an ordinary user?
 - A. Either root or the user, with the user's settings taking precedence
 - B. Either root or the user, with root's settings taking precedence
 - C. root only
 - D. The user only
3. Where is the best location for the current directory indicator (`.`) to reside in root's `PATH` environment variable?
 - A. Before all other directories
 - B. After all other directories
 - C. Nowhere; it shouldn't be in root's path
 - D. Wherever is convenient
4. After using a text editor to create a shell script, what step should you ordinarily take before using the script?
 - A. Set one or more executable bits using `chmod`.
 - B. Copy the script to the `/usr/bin/scripts` directory.
 - C. Compile the script by typing `bash scriptname`, where *scriptname* is the script's name.
 - D. Run a virus checker on the script to be sure it contains no viruses.
5. Which of the following wildcards will match the filenames `scan.txt`, `Skin.txt`, and `skin.txt` but not `scan.odt`, `skin.TXT`, or `skinny.txt`?
 - A. `*.*`
 - B. `s*.txt`
 - C. `[Ss]*n.txt`
 - D. `s??n*`

6. To simplify access to a directory with a long path, `/opt/clipart/standard/office/products`, you want to create a link. You type `ln /opt/clipart/standard/office/products ./ca`, but you receive an error message. How must you change this command to make it work?
- A. Change `ln` to `mv`.
 - B. Change `./ca` to `./products`.
 - C. Add `-s` after `ln`.
 - D. Add `/` after `products`, with no intervening space.
7. Which of the following procedures normally launches a shell? (Choose all that apply.)
- A. Starting an `xterm` window.
 - B. Typing `shell` at a command prompt.
 - C. Logging in using SSH.
 - D. You can't; the shell is started automatically at boot time.
8. What key does the Bash shell use to complete filenames based on the first few characters?
- A. End
 - B. Tab
 - C. Enter
 - D. Insert
9. What command would you type to change the ownership of `somefile.txt` from `ralph` to `tony`?
- A. `chown ralph:tony somefile.txt`
 - B. `chmod somefile.txt tony`
 - C. `chown somefile.txt tony`
 - D. `chown tony somefile.txt`
10. Which of the following `umask` values will result in files with `rw-r-----` permissions?
- A. 640
 - B. 210
 - C. 022
 - D. 027
11. You want to discover the sizes of several dot files in a directory. Which of the following commands might you type to do this?
- A. `ls -la`
 - B. `ls -p`
 - C. `ls -R`
 - D. `ls -d`

12. You want to move a file from your hard disk to a floppy disk. Which of the following is true?
 - A. You'll have to use the `--preserve` option to `mv` to keep ownership and permissions set correctly.
 - B. The `mv` command will adjust filesystem pointers without physically rewriting data if the floppy uses the same filesystem type as the hard disk partition.
 - C. You must use the same filesystem type on both media to preserve ownership and permissions.
 - D. The `mv` command will delete the file on the hard disk after copying it to the floppy.
13. You type `mkdir one/two/three` and receive an error message that reads, in part, `No such file or directory`. What can you do to overcome this problem? (Choose all that apply.)
 - A. Add the `--parents` parameter to the `mkdir` command.
 - B. Issue three separate `mkdir` commands: `mkdir one`, then `mkdir one/two`, then `mkdir one/two/three`.
 - C. Type `touch /bin/mkdir` to be sure the `mkdir` program file exists.
 - D. Type `rmdir one` to clear away the interfering base of the desired new directory tree.
14. A program you're using logs errors to the `~/someprog/errors.txt` file. You've seen cryptic error messages in this file, and you want to determine what function of the program is generating these messages. What command would you type to enable you to see these messages as they're written to the file?
 - A. `tail -f ~/someprog/errors.txt`
 - B. `vi ~/someprog/errors.txt`
 - C. `chmod 644 ~/someprog/errors.txt`
 - D. `monitor-logs`
15. You want to learn about the `/dev/tty` device file, so you type `man tty`. This action, however, produces information on the `tty` command, which displays the name of the terminal device from which it was called. How would you obtain the documentation you wanted?
 - A. Type `man /dev/tty`.
 - B. Type `man 2 tty`.
 - C. Type `man 4 tty`.
 - D. Type `man tty device`.
16. Which of the following file-location commands is likely to take the *most* time to find a file that might be located anywhere on the computer?
 - A. The `find` command.
 - B. The `locate` command.
 - C. The `whereis` command.
 - D. They're all equal in speed.

17. Which of the following commands is an improved version of `more`?
- A. `grep`
 - B. `tail`
 - C. `cat`
 - D. `less`
18. You want to find the documentation for a Linux scanner program, but you can't remember the name of the software. What command might you type to help you track down the documentation?
- A. `info scanner`
 - B. `help scanner`
 - C. `apropos scanner`
 - D. `find scanner`
19. Which of the following commands would you type to change the group associated with the `modes.tex` file to `marketing`?
- A. `chgrp modes.tex marketing`
 - B. `chgrp marketing modes.tex`
 - C. `group modes.tex marketing`
 - D. `newgrp modes.tex marketing`
20. Which of the following commands would you type to print lines from the file `world.txt` that contain matches to `changes` and `changed`?
- A. `grep change[ds] world.txt`
 - B. `sed change[d-s] world.txt`
 - C. `find "change'd|s'" world.txt`
 - D. `search world.txt changes changed`

Answers to Review Questions

1. C. Option A sets the path to contain *only* the `/usr/local/bigprog/bin` directory, rather than *adding* that directory to the existing path. Options B and D use the `tcsh` syntax for setting the path, and option B uses it incorrectly (`/etc/profile` is used for setting environment variables in Bash, not `tcsh`).
2. A. The `root` user may set environment variables in `/etc/profile` or other system-wide configuration files, and users may set their own environment variables in `.bashrc` or other user-level configuration files or by typing them in manually. Because the user's settings come later, they override system defaults, if in conflict.
3. C. The current directory indicator is particularly dangerous in `root`'s `PATH` environment variable because it can be used by unscrupulous local users to trick `root` into running programs of the unscrupulous user's design.
4. A. Scripts, like binary programs, normally have at least one executable bit set, although they can be run in certain ways without this feature. There is no standard `/usr/bin/scripts` directory, and scripts can reside in any directory. Scripts are interpreted programs, which means they don't need to be compiled. Typing **`bash scriptname`** will run the script. Viruses are extremely rare in Linux, and because you just created the script, the only ways it could possibly contain a virus would be if your system were already infected or if you wrote it as a virus.
5. C. The `[Ss]` matching set matches both the uppercase `S` in `Skin.txt` and the lowercase `s` in `scan.txt` and `skin.txt`. This is necessary to match all of the required filenames, but it also matches all of the unwanted filenames. The following `*` matches any number of intervening characters. The trailing `n.txt` matches the ends of all three of the target filenames but does not match any of the unwanted filenames. Thus, option C is correct. Option A matches all of the files (both wanted and unwanted). Option B doesn't match `Skin.txt` and it does match `skinny.txt`; thus, it won't do. Because option D begins with a lowercase `s`, it doesn't match `Skin.txt`. This option also incorrectly matches all of the unwanted filenames.
6. C. By default, `ln` creates hard links, but most filesystems don't permit hard links between subdirectories. Furthermore, if the current directory and the target directory are on different low-level filesystems, you can't create hard links between them. Thus, the command will likely fail. The `-s` option to `ln`, as specified in option C, creates a symbolic link rather than a hard link. Since symbolic links to directories and across filesystems are legal, this command is likely to succeed. Changing `ln` to `mv`, as in option A, turns the command into an attempt to move the directory, which might or might not work, but is definitely not what's desired. Option B's change will have no effect; links need not have the same name as the linked-to file or directory. Option D is an attempt to explicitly define the linked-to directory as a directory. It will also have no effect.

7. A, C. Shells are started automatically when you log in or start `xterm` windows unless you configure your account strangely or specify another program to run when you launch an `xterm`. Typing `shell` won't start a shell, because no standard shell is called `shell`. (Typing the shell name will do the job, though.) Shells aren't normally started when the computer boots; you must first log in.
8. B. When you press the Tab key when you are typing a command or filename, Bash checks to see whether the characters you've typed so far are enough to uniquely identify the command or filename. If they are, Bash completes the command or filename, saving you keystrokes.
9. D. Typing `chown ralph:tony somefile.txt` sets the owner of the file to `ralph` and the group to `tony`. The `chmod` command is used to change file permissions, not ownership. Option C reverses the order of the filename and the owner. Answer D uses the correct command and options.
10. D. Option D, 027, removes write permissions for the group and all world permissions. (Files normally don't have execute permissions set, but explicitly removing write permissions when removing read permissions ensures reasonable behavior for directories.) Option A, 640, is the octal equivalent of the desired `rw-r-----` permissions, but the `umask` sets the bits that are to be *removed* from permissions, not those that are to be set. Option B, 210, would remove write permission for the owner, but it would not remove write permission for the group, which is incorrect. This would also leave all world permissions open. Finally, option C, 022, would not remove world read permission.
11. A. The `-l` parameter produces a long listing, including file sizes. The `-a` parameter produces a listing of all files in a directory, including the dot files. Combining the two produces the desired information (along with information on other files).
12. D. When moving from one partition or disk to another, `mv` must necessarily read and copy the file and then delete the original if that copy was successful. If both filesystems support ownership and permissions, they'll be preserved; `mv` doesn't need an explicit `--preserve` option to do this, and this preservation does not rely on having exactly the same filesystem types. Although `mv` doesn't physically rewrite data when moving within a single low-level filesystem, this approach cannot work when you are copying to a separate low-level filesystem (such as from a hard disk to a floppy disk); if the data isn't written to the new location, it won't be accessible should the disk be inserted in another computer.
13. A, B. If you try to create a directory inside a directory that doesn't exist, `mkdir` responds with a `No such file or directory` error. The `--parents` parameter tells `mkdir` to automatically create all necessary parent directories in such situations. You can also manually do this by creating each necessary directory separately. (It's possible that `mkdir one` wouldn't be necessary in this example, if the directory `one` already exists. No harm will come from trying to create a directory that already exists, although `mkdir` will return a `File exists` error.)
14. A. The `tail` command's `-f` (or `--follow`) option sets it to monitor a file and display additions to the file as they're made. Thus, option A does the specified job. The `vi` command is a text editor, and `chmod` changes permissions on a file; thus, neither option B nor C will do what's wanted. Option D's `monitor-logs` command is fictitious.

15. C. The Linux man page system is divided into numbered sections, and section 4 is devoted to device files. Specifying the section number before the search term restricts the search to that section, so option C will produce the desired documentation, if it's available. Specifying the complete pathname to the device, as in option A, will most likely produce an error. Option B specifies the incorrect section number of 2 (system calls). Option D will look up the default (incorrect) `tty` entry and then attempt to find an entry for `device`.
16. A. The `find` utility operates by searching all files in a directory tree, so it is likely to take a long time to search all a computer's directories. The `locate` program uses a precompiled database, and `whereis` searches a limited set of directories, so these commands will take less time.
17. D. The `less` program, like `more`, displays a text file a page at a time. The `less` utility also includes the ability to page backward in the text file, search its contents, and more.
18. C. The `apropos` command searches through certain sections of man pages to find the specified keyword and returns the names of the man pages in which that keyword was found. The `info` command of option A is an alternative to `man`, so unless your system happens to have a program called *scanner*, it's unlikely to help. There is no standard Linux command called `help`, so option B is also incorrect. Although the `find` command could conceivably be used to help find scanner-related tools, the command specified in option D searches only the current directory for a file called `scanner`, so it won't help.
19. B. The `chgrp` utility is used to change the group associated with a file, just as `chown` is used to change the owner associated with the file. The correct syntax requires the first parameter given be the name of the group to be associated with the file, followed by the name of the file. There is no `group` utility, and `newgrp` does not perform this function.
20. A. The `grep` utility is used to find matching text within a file and print those lines. It accepts regular expressions, which allow for the placing of the two characters you are looking for within brackets. The syntax for `sed` and `find` would not perform the needed task, and there is no standard Linux utility named `search`.



Chapter 3

Managing Processes and Editing Files

THE FOLLOWING COMPTIA OBJECTIVES ARE COVERED IN THIS CHAPTER:

- ✓ **2.1** Given a scenario, use the following fundamental Linux tools, techniques and resources (**File editing with Vi**; **Process management**: ps, kill, top, iostat, pstree, nice, renice, **signals**, **PID**, **PPID**; **Kernel/architecture information**: cat, /proc/version, uname, **common sysctl settings**, /etc/sysctl.conf)
- ✓ **2.8** Implement task scheduling using the following tools (**cron**: cron.allow, cron.deny, crontab **command syntax**, crontab **file format**, at: atq)
- ✓ **2.9** Utilize performance monitoring tools and concepts to identify common problems (**Commands**: sar, iostat, vmstat, uptime, top; **Load average**)



A running Linux system consists largely of processes—that is, programs that are active. Linux provides tools that enable you to view, start, stop, change the priority of, and otherwise manipulate processes. The ability to do this enables you to keep the computer running smoothly and to correct problems that can be caused by processes that misbehave.

In addition to the basic process manipulation tools, this chapter covers timed process activation, both on an ongoing basis (running a program once a week, for instance) and on a one-time basis (running a program once on next Thursday, for instance). The *kernel* is arguably the most important Linux process of all, because the kernel is the software component that interfaces between the hardware and all other software. The kernel also controls other processes by allocating memory, assigning CPU time, and so on.

This chapter concludes with a look at editing text files. Although many text editors are available in Linux, the Linux+ objectives mention only one: Vi. This editor is a simple but powerful editor. Most users find it strange, so it requires some learning to master.



Objective 2.1 is covered partly in this chapter and partly in Chapter 2, “Using Text-Mode Commands.”

Managing Processes

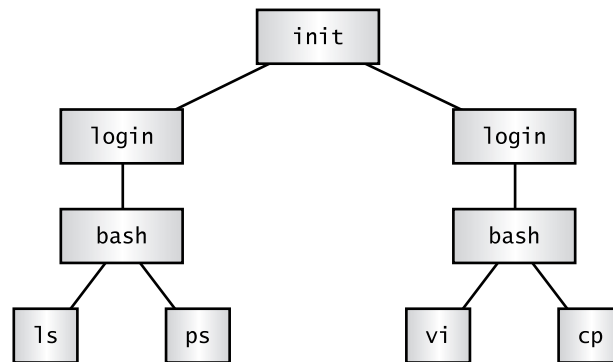
For the most part, files are fairly static; once created, a file is likely to remain unchanged for minutes, hours, days, or longer. A running program, or *process*, is a much more dynamic entity; it’s stored in memory and directs the computer’s CPU to perform some task. Because of this fact, processes present different management challenges than do files.

Understanding Processes

Processes are started in various ways. These include being launched from a shell, as described in Chapter 2, and being started from a system startup script, as described in more detail in Chapter 4, “Managing System Services.” However a process is started, it can be controlled in various ways. You can obtain lists of running processes, restrict their CPU use, and even kill a process you don’t want to have running.

Before proceeding, it's important that you understand a bit of terminology. In Linux, a process is more or less synonymous with a running program. Because Linux is a multiuser, multitasking OS, it's possible for one program to be running as more than one process at a time. For instance, suppose that `tbaker` and `smccoy` both use `Vi` to edit text files. The computer will have two `Vi` processes running at once. Indeed, a single user can do this. It's also possible for a single program to create (or *fork*) subprocesses. For instance, `Vi` can launch a spell checker program. In fact, this is what happens when you launch a program from a shell—the shell forks the program you're launching. When one process forks another, the original process is known as the *parent process*, and the forked process is known as the *child process*. This parent/child relationship produces a tree-like hierarchy that ultimately leads back to `init`, the first process run by the kernel when it boots. Figure 3.1 shows a simplified example. In Figure 3.1, `init` forks the `login` processes, which in turn fork `bash` processes, which fork additional processes. (It's actually slightly more complex than this; `init` doesn't directly fork `login` but instead does this by using another process, such as `getty`.) This can continue for an arbitrary number of layers, although many programs aren't able to fork others.

FIGURE 3.1 Linux processes have parents, leading back to `init`, the first program the Linux kernel runs.



Examining Process Lists with `ps`

One of the most important tools in process management is `ps`. This program displays processes' status (which is why it's called `ps`). It sports many useful options, and it's helpful in monitoring what's happening on a system. This can be particularly critical when the computer isn't working as it should be—for instance, if it's unusually slow. The `ps` program supports an unusual number of options, but just a few of them will take you a long way. Likewise, interpreting `ps` output can be tricky because so many options modify what's available. Some `ps`-like programs, most notably `top`, also deserve some attention.

Examining *ps* Options

The official syntax for *ps* is fairly simple:

```
ps [options]
```

This simplicity of form hides considerable complexity because *ps* supports three different *types* of options, as well as many options within each type. The three types of options are as follows:

Unix98 options These single-character options may be grouped together and are preceded by a single dash (-).

BSD options These single-character options may be grouped together and must *not* be preceded by a dash.

GNU long options These multicharacter options are not grouped together. They're preceded by two dashes (--).

Options that may be grouped together may be clustered without spaces between them. For instance, rather than typing ***ps -a -f***, you can type ***ps -af***. The reason for so much complexity is that the *ps* utility has historically varied a lot from one Unix OS to another. The version of *ps* that ships with major Linux distributions attempts to implement most features from all these different *ps* versions, so it supports many different personalities. In fact, you can change some of its default behaviors by setting the `PS_PERSONALITY` environment variable to `posix`, `old`, `linux`, `bsd`, `sun`, `digital`, or various others. (Chapter 2 describes how to set environment variables.) The rest of this section describes the default *ps* behavior on most Linux systems. Table 3.1 summarizes some of the more useful *ps* features.

TABLE 3.1 Common *ps* Options

Unix98 Option(s)	BSD Option(s)	GNU Long Option(s)	Description
		--help	This option displays a summary of some of the more common <i>ps</i> options.
-A, -e	x		By default, <i>ps</i> displays only processes that were run from its own terminal (xterm, text-mode login, or remote login). The -A and -e options cause it to display all the processes on the system, and x displays all processes owned by the user who gives the command. The x option also increases the amount of information that's displayed about each process.

TABLE 3.1 Common `ps` Options (*continued*)

Unix98 Option(s)	BSD Option(s)	GNU Long Option(s)	Description
<code>-u user</code>	<code>U user</code>	<code>--User user</code>	You can obtain a list of processes owned by a specified user with this option. The <i>user</i> variable may be a username or a user ID (UID) number.
<code>-f, -l</code>	<code>j, l, u, v</code>		These options display extra information. The details of what is displayed vary with the specific option used.
<code>-H, -f</code>		<code>--forest</code>	These options group processes and use indentation to show the hierarchy of relationships between processes. They're useful if you're trying to trace the parentage of a process.
<code>-w</code>	<code>w</code>		The <code>ps</code> command output can be more than 80 columns wide. Normally, <code>ps</code> truncates its output so that it will fit on your screen or <code>xterm</code> . The <code>-w</code> and <code>w</code> options tell <code>ps</code> not to do this, which can be useful if you direct the output to a file, as in <code>ps w > ps.txt</code> . You can then examine the output file in a text editor that supports wide lines.

You can combine these `ps` options in many ways to produce the output you want. You'll probably need to experiment to learn which options produce the desired results because each of these options modifies the output in some way. Even those that would seem to influence just the selection of processes to list sometimes modify the information that's provided about each process.

Interpreting `ps` Output

Listing 3.1 and Listing 3.2 show a couple of examples of `ps` in action. Listing 3.1 shows the output from typing `ps -u rodsmith --forest`, and Listing 3.2 shows the output from typing `ps u U rodsmith`.

Listing 3.1: Output of `ps -u rodsmith --forest`

```
$ ps -u rodsmith --forest
  PID TTY          TIME CMD
 2451 pts/3    00:00:00 bash
 2551 pts/3    00:00:00 ps
 2496 ?          00:00:00 kvt
 2498 pts/1    00:00:00 bash
 2505 pts/1    00:00:00  \_ nedit
 2506 ?          00:00:00    \_ csh
 2544 ?          00:00:00      \_ xeyes
19221 ?          00:00:01 dfm
```

Listing 3.2: Output of `ps u U rodsmith`

```
$ ps u U rodsmith
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
rodsmith 19221  0.0  1.5  4484 1984 ?        S    May07    0:01 dfm
rodsmith 2451  0.0  0.8  1856 1048 pts/3    S    16:13    0:00 -bash
rodsmith 2496  0.2  3.2  6232 4124 ?        S    16:17    0:00 /opt/kd
rodsmith 2498  0.0  0.8  1860 1044 pts/1    S    16:17    0:00 bash
rodsmith 2505  0.1  2.6  4784 3332 pts/1    S    16:17    0:00 nedit
rodsmith 2506  0.0  0.7  2124 1012 ?        S    16:17    0:00 /bin/cs
rodsmith 2544  0.0  1.0  2576 1360 ?        S    16:17    0:00 xeyes
rodsmith 2556  0.0  0.7  2588  916 pts/3    R    16:18    0:00 ps u U
```

The output produced by `ps` normally begins with a heading line, which displays the meaning of each column. Important information that might be displayed (and labeled) includes the following:

Username The name of the user who runs the programs. Listings 3.1 and 3.2 restricted this output to one user to limit the size of the listings.

Process ID The process ID (PID) is a number that's associated with the process. This item is particularly important because you need it to modify or kill the process, as described later in this chapter.

Parent process ID The parent process ID (PPID) identifies the process's parent. (Neither Listing 3.1 nor Listing 3.2 shows the PPID, though.)

TTY The teletype (TTY) is a code used to identify a terminal. As illustrated by Listings 3.1 and 3.2, not all processes have TTY numbers—X programs and daemons, for instance, do not. Text-mode programs do have these numbers, though, which point to a console, `xterm`, or remote login session.

CPU time The `TIME` and `%CPU` headings are two measures of CPU time used. The first indicates the total amount of CPU time consumed, and the second represents the percentage

of CPU time the process is using when `ps` executes. Both can help you spot runaway processes—those that are consuming too much CPU time. Unfortunately, just what constitutes “too much” varies from one program to another, so it’s impossible to give a simple rule to help you spot a runaway process.

CPU priority As described shortly, in “Restricting Processes’ CPU Use,” it’s possible to give different processes different priorities for CPU time. The `NI` column, if present (it’s not in the preceding examples) lists these priority codes. The default value is 0. Positive values represent *reduced* priority, while negative values represent *increased* priority.

Memory use Various headings indicate memory use—for instance, `RSS` is resident set size (the memory used by the program and its data), and `%MEM` is the percentage of memory the program is using. Some output formats also include a `SHARE` column, which is memory that’s shared with other processes (such as shared libraries). As with CPU use measures, these columns can help point you to the sources of difficulties, but because legitimate memory needs of programs vary so much, it’s impossible to give a simple criterion for when a problem exists.

Command The final column in most listings is the command used to launch the process. This is truncated in Listing 3.2 because this format lists the complete command, but so much other information appears that the complete command won’t usually fit on one line. (This is where the wide-column options can come in handy.)

As you can see, a lot of information can be gleaned from a `ps` listing—or perhaps that should be the plural *listings*, because no one format includes all of the available information. For the most part, the `PID`, username, and command are the most important pieces of information. In some cases, though, you may need specific other components. If your system’s memory or CPU use has skyrocketed, for instance, you’ll want to pay attention to the memory or CPU use columns.



It’s often necessary to find specific processes. You might want to find the `PID` associated with a particular command in order to kill it, for instance. This information can be gleaned by piping the `ps` output through `grep`, as in `ps ax | grep bash` to find all the instances of Bash.

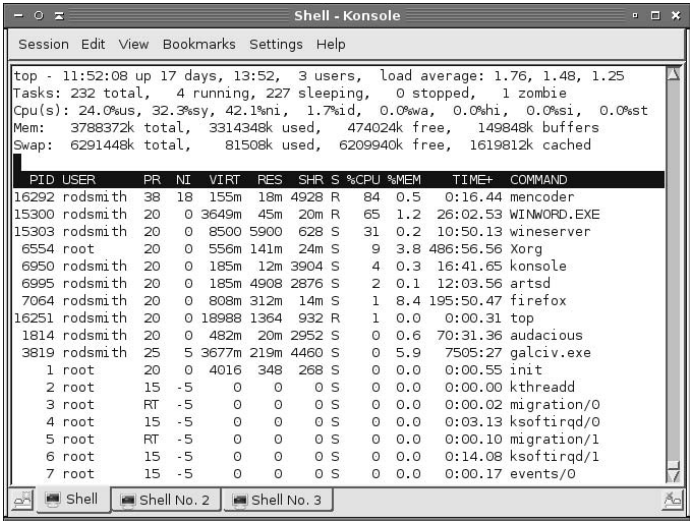
Although you may need a wide screen or `xterm` to view the output, you may find `ps -A --forest` to be a helpful command in learning about your system. (The `pstree` command produces a similar display.) Processes that don’t fall off others were either started directly by `init` or have had their parents killed, and so they have been “adopted” by `init`. Most of these processes are fairly important—they’re servers, login tools, and so on. Processes that hang off several others in this tree view, such as `xeyes` and `nedit` in Listing 3.1, are mostly user programs launched from shells.

Viewing Processes Dynamically

If you want to know how much CPU time various processes are consuming relative to one another, or if you simply want to quickly discover which processes are consuming the most CPU time, a tool called `top` is the one for the job. The `top` tool is a text-mode program,

but of course it can be run in an xterm, as shown in Figure 3.2, and there are also GUI variants, like `kpm` and `gnome-system-monitor`. By default, `top` sorts its entries by CPU use, and it updates its display every few seconds. This makes it a very good tool for spotting runaway processes on an otherwise lightly loaded system—those processes almost always appear in the first position or two, and they consume an inordinate amount of CPU time. By looking at Figure 3.2, you might think that `mencoder` is such a process, but in fact, it’s legitimately consuming a lot of CPU time. You’ll need to be familiar with the purposes and normal habits of programs running on *your* system in order to make such determinations; the legitimate needs of different programs vary so much that it’s impossible to give a simple rule for judging when a process is consuming too much CPU time.

FIGURE 3.2 The `top` command shows system summary information and information on the most CPU-intensive processes on a computer.



Like many Linux commands, `top` accepts several options. Table 3.2 summarizes the most useful of these options.

TABLE 3.2 Common `top` Options

Option Name	Meaning
<code>-d delay</code>	This option specifies the delay between updates, which is normally 5 seconds.
<code>-p pid</code>	If you want to monitor specific processes, you can list them using this option. You’ll need the PIDs, which you can obtain with <code>ps</code> , as described earlier. You can specify up to 20 PIDs by using this option multiple times, once for each PID.

TABLE 3.2 Common top Options (*continued*)

Option Name	Meaning
-n <i>iter</i>	You can tell top to display a certain number of updates (<i>iter</i>) and then quit. (Normally, top continues updating until you terminate the program.)
-b	This option specifies batch mode, in which top doesn't use the normal screen update commands. You might use this to log CPU use of targeted programs to a file, for instance.

You can do more with top than watch it update its display. When it's running, you can enter any of several single-letter commands, some of which prompt you for additional information. Table 3.3 summarizes these commands.

TABLE 3.3 Common top Interactive Commands

Option Name	Meaning
h or ?	These keystrokes display help information.
k	You can kill (terminate) a process with this command. The top program will ask for a PID number and signal number, and if it's able to kill it, it will do so. (The upcoming section "Killing Processes" describes other ways to kill processes.)
q	This option quits from top.
r	You can change a process's priority with this command. You'll have to enter the PID number and a new priority value—a positive value will decrease its priority, and a negative value will increase its priority, assuming it has the default 0 priority to begin with. Only root may increase a process's priority. The <code>renice</code> command (described shortly, in "Restricting Processes' CPU Use") is another way to accomplish this task.
s	This command changes the display's update rate, which you'll be asked to enter (in seconds).
p	This sets the display to sort by CPU usage, which is the default.
M	You can change the display to sort by memory usage with this command.

More commands are available in `top` (both command-line options and interactive commands) than can be summarized here; consult the `top` man page for more information.

One of the pieces of information provided by `top` is the *load average*, which is a measure of the demand for CPU time by applications. In Figure 3.2, you'll see three load-average estimates on the top line; these correspond to the load average in the previous one minute, five minutes, and 15 minutes. A system on which no programs are demanding CPU time will have a load average of 0. A system with one program running CPU-intensive tasks will have a load average of 1.0. Higher load averages reflect programs competing for available CPU time. You can also find the current load average via the `uptime` command, which displays the load average along with information on how long the computer has been running. The load average can be useful in detecting runaway processes. For instance, if a system normally has a load average of 0.5 but it suddenly gets stuck at a load average of 2.5, there may be a couple of CPU-hogging processes that have *hung*—that is, become unresponsive. Hung processes sometimes needlessly consume a lot of CPU time. You can use `top` to locate these processes and, if necessary, kill them.

You might think that a load average of 1.0 or below is optimal, since a higher load average means that the computer must divide its attention between competing processes. This is sometimes true, but sometimes it's not. Modern computers often have two or more CPUs (or CPU cores), which means that the system can divide its CPU time between as many processes as it has cores with almost no slowdown. For instance, on a dual-core system, a load average of up to 2.0 will produce almost no speed reduction. Furthermore, modern CPUs are fast enough that the slowdown caused by the division of a CPU's time between processes may be unimportant. Using `nice` (described in the next section) can interact with this factor. This effect is hard to quantify, though. Sometimes processes competing for CPU time will cause noticeable slowdown, as in the slow completion of a lengthy multimedia encoding job. Other times the slowdown will cause a barely noticeable effect, as in a fraction-of-a-second difference in the time required to display a program's menu bar. Some Linux systems are intended to be run with greater-than-1.0 load averages, but with CPUs that are fast enough that the slowdown caused by these high load averages will be unimportant. This might be true of a multiuser server, for instance.

Restricting Processes' CPU Use

There may be times when you'll want to prioritize your programs' CPU use. For instance, you might be running a program that's very CPU-intensive but that will take a long time to finish its work, and you don't want that program to interfere with others that are of a more interactive nature. Alternatively, on a heavily loaded computer, you might have a job that's more important than others that are running, so you might want to give it a priority boost. In either case, the usual method of accomplishing this goal is through the `nice` and `renice` commands. You can use `nice` to launch a program with a specified priority or use `renice` to alter the priority of a running program.

You can assign a priority to `nice` in any of three ways: by specifying the priority preceded by a dash (this works well for positive priorities but makes them look like negative priorities), by specifying the priority after a `-n` parameter, or by specifying the priority after

an `--adjustment=` parameter. In all cases, these parameters are followed by the name of the program you want to run:

```
nice [argument] [command [command-arguments]]
```

For instance, the following three commands are all equivalent:

```
$ nice -12 number-crunch data.txt
$ nice -n 12 number-crunch data.txt
$ nice --adjustment=12 number-crunch data.txt
```

All three of these commands run the `number-crunch` program at priority 12 and pass it the `data.txt` file. If you omit the adjustment value, `nice` uses 10 as a default. The range of possible values is `-20` to `19`, with negative values having the highest priority. Only `root` may launch a program with increased priority (that is, give a negative priority value), but any user may use `nice` to launch a program with low priority. The default priority for a program run without `nice` is `0`.

If you've found that a running process is consuming too much CPU time or is being swamped by other programs and so should be given more CPU time, you can use the `renice` program to alter its priority without disrupting the program's operation. The syntax for `renice` is as follows:

```
renice priority [[-p] pids] [[-g] pgrps] [[-u] users]
```

You must specify the *priority*, which takes the same values as with `nice`. In addition, you must specify one or more PIDs (*pids*), one or more group IDs (*pgrps*), or one or more usernames (*users*). In the latter two cases, `renice` changes the priority of all programs that match the specified criterion—but only `root` may use `renice` in this way. Also, only `root` may increase a process's priority. If you give a numeric value without a `-p`, `-g`, or `-u` option, `renice` assumes the value is a PID. You may mix and match these methods of specification. For instance, you might enter the following command:

```
# renice 7 16580 -u pdavison tbaker
```

This command sets the priority to `7` for PID `16580` and for all processes owned by `pdavison` and `tbaker`.

Killing Processes

Sometimes reducing a process's priority isn't a strong enough action. A program may have become totally unresponsive, or you might want to terminate a process that shouldn't be running at all. In these cases, the `kill` command is the tool to use. This program sends a *signal* (a method that Linux uses to communicate with processes) to a process. The signal is usually sent by the kernel, the user, or the program itself to terminate the process. Linux supports many numbered signals, each of which is associated with a specific name. You can see them all by typing `kill -l`. If you don't use `-l`, the syntax for `kill` is as follows:

```
kill -s signal pid
```



Although Linux includes a `kill` program, many shells, including Bash and `csh`, include built-in `kill` equivalents that work in much the same way as the external program. If you want to be sure you're using the external program, type its complete path, as in `/bin/kill`.

The `-s signal` parameter sends the specified signal to the process. You can specify the signal using either a number (such as 9) or a name (such as `SIGKILL`). The signals you're most likely to use are 1 (`SIGHUP`, which causes many server programs to reread their configuration files), 9 (`SIGKILL`, which causes the process to exit without performing routine shutdown tasks), and 15 (`SIGTERM`, which causes the process to exit but allows it to close open files and so on). If you don't specify a signal, the default is 15 (`SIGTERM`). You can also use the shortened form `-signal`. If you do this and use a signal name, you should omit the `SIG` portion of the name—for instance, use `KILL` rather than `SIGKILL`. The `pid` option is, of course, the PID for the process you want to kill. You can obtain this number from `ps` or `top`.



The `kill` program will kill only those processes owned by the user who runs `kill`. The exception is if that user is `root`; the superuser may kill any user's processes.

A variant on `kill` is `killall`, which has the following form:

```
killall [options] [--] name [...]
```

This command kills a process based on its name rather than its PID number. For instance, `killall vi` kills all the running processes called `vi`. You may specify a signal in the shortened form (`-signal`) or by preceding the signal number with `-s` or `--signal`. As with `kill`, the default is 15 (`SIGTERM`). One potentially important option to `killall` is `-i`, which causes it to ask for confirmation before sending the signal to each process. You might use it like this:

```
$ killall -i vi
Kill vi(13211) ? (y/n) y
Kill vi(13217) ? (y/n) n
```

In this example, two instances of the Vi editor were running, but only one should have been killed. As a general rule, if you run `killall` as `root`, you should use the `-i` parameter; if you don't, it's all too likely that you'll kill processes that you should not, particularly if the computer is being used by many people at once.



Some versions of Unix provide a `killall` command that works very differently from Linux's `killall`. This alternate `killall` kills all the processes started by the user who runs the command. This is a potentially much more destructive command, so if you ever find yourself on a non-Linux system, *do not* use `killall` until you've discovered what that system's `killall` does, say by reading the `killall` man page.

Controlling Foreground and Background Processes

Less extreme process management tools enable you to control whether a process is running in the foreground or the background—that is, whether or not it’s monopolizing the use of the terminal from which it was launched. Normally, when you launch a program, it takes over the terminal, preventing you from doing other work in that terminal. (Some programs, though, release the terminal. This is most common for servers and some GUI programs.)

If a program is running but you decide you want to use that terminal for something else, pressing Ctrl+Z normally pauses the program and gives you control of the terminal. (An important point is that this procedure pauses the program, so if it’s performing real work, that work stops!) This can be handy if, say, you’re running a text editor in a text-mode login and you want to check a filename so you can mention it in the file you’re editing. You’d press Ctrl+Z and type **ls** to get the file listing. To get back to the text editor, you’d then type **fg**, which restores the text editor to the foreground of your terminal. If you’ve suspended several processes, you’d add a job number, as in **fg 2** to restore job 2. You can obtain a list of jobs associated with a terminal by typing **jobs**, which displays the jobs and their job numbers.



Job numbers are not the same as PIDs. PIDs are used by the kernel to track processes, and many utilities, such as **ps**, **top**, and **kill**, report PIDs or use them. Job numbers are linked to the terminal from which the process was launched and are used by fewer programs. Don’t try to use a PID in place of a job number, or vice versa.

A variant on **fg** is **bg**. Where **fg** restores a job to the foreground, **bg** restores a job to running status, but in the background. You might use this command if the process you’re running is performing a CPU-intensive task that requires no human interaction but you want to use the terminal in the meantime. Another use of **bg** is in a GUI environment—after launching a GUI program from an **xterm** or similar window, that shell is tied up servicing the GUI program, which probably doesn’t really need the shell. Pressing Ctrl+Z in the **xterm** window will enable you to type shell commands again, but the GUI program will be frozen. To unfreeze the GUI program, type **bg** in the shell, which enables the GUI program to run in the background while the shell continues to process your commands.

An alternative to launching a program, using Ctrl+Z, and typing **bg** is to append an ampersand (&) to the command when launching the program. For instance, rather than edit a file with the NEdit GUI editor by typing **nedit myfile.txt**, you could type **nedit myfile.txt &**. This command launches the **nedit** program in the background from the start, leaving you able to control your **xterm** window for other tasks.

Monitoring System Statistics

In addition to **ps**, **top**, and similar tools for checking on specific processes, several tools help you to monitor the health of the system as a whole. In particular, **sar**, **iostat**, and **vmstat** all provide overall system statistics.



The `sar` tool requires that you launch the `sysstat` SysV startup script. Only activities that occur after this script is launched will be recorded.

Obtaining General-Purpose System Statistics

The `sar` utility is extremely powerful; it can report on a wide range of system use measures over time. Its basic format is as follows:

`sar [options] [interval [count]]`

interval is the interval between measurements, in seconds; and *count* is the number of measurements to report. Table 3.4 summarizes some of the more useful `sar` options; however, there are many more. You should consult the program's man page for details. If you don't pass any *options* to `sar`, it produces a CPU utilization report. If you omit the *interval* and *count* options, `sar` produces information on recent activity; otherwise, it waits and displays information on the next few time periods, as specified by those options.

TABLE 3.4 Common `sar` Options

Option Name	Meaning
-A	Summarizes a wide variety of information.
-b	Displays I/O and transfer rate information.
-B	Summarizes paging (virtual memory) statistics.
-c	Displays information on process creation—how many processes are started per second.
-d	Reports on block device activity.
-e [hh:mm:ss]	Sets the ending time for the report; without this option, the program continues reporting indefinitely.
-i interval	Reports data as close as possible to the specified <i>interval</i> (in seconds); used when reporting already-recorded data.
-I irq	Reports on use of the specified interrupt (IRQ).
-n code	Reports on network use. The <i>code</i> keyword can be DEV, NFS, or various other values to report on specific types of network activity. Consult the man page for details.

TABLE 3.4 Common `sar` Options (*continued*)

Option Name	Meaning
<code>-o filename</code>	Saves data to the specified file in binary form.
<code>-P cpu</code>	Displays CPU use data. The <i>cpu</i> code may be a CPU number or ALL.
<code>-r</code>	Reports memory and swap space use information. This information is more detailed than that reported by the <code>free</code> utility.
<code>-s [hh:mm:ss]</code>	Sets the start time for data display.
<code>-u</code>	Reports CPU use data. This is the default.
<code>-W</code>	Displays swap statistics.

The `sar` utility is useful for tracking down the source of system slowdowns. If you receive complaints that the computer slows down at particular times of the day, you can use `sar` to examine the load on memory, CPU, devices, and so on, throughout the day to look for patterns of what might be causing the slowdown. The tool is most helpful if you're familiar with normal operations, so you may want to use it on a regular basis for a while, preferably on several Linux systems that perform different types of tasks, to see what sorts of loads occur on a variety of Linux systems.

Obtaining Input/Output Statistics

The `iostat` utility is more specialized than `sar`, but it can still produce a wide range of system statistics. These include reports on CPU utilization, block device input/output, and Network File System (NFS) activity. Its syntax is similar to that of `sar`:

```
iostat [options] [interval [count]]
```

Table 3.5 summarizes some of the more common `iostat` options. If used without options, `iostat` produces a CPU report followed by a report on block devices, similar to what typing `iostat -cd` produces.

TABLE 3.5 Common `iostat` Options

Option Name	Meaning
<code>-c</code>	Creates a CPU utilization report
<code>-d</code>	Creates a device utilization report

TABLE 3.5 Common iostat Options *(continued)*

Option Name	Meaning
-k	Displays statistics in kilobytes per second rather than blocks per second
-m	Displays statistics in megabytes per second rather than blocks per second
-n	Creates an NFS utilization report

You can use `iostat` for many of the same purposes as you'd use `sar`—to track down the sources of problems when they're reported. As with `sar`, you'll find this task easier if you're familiar with the usual output of `iostat`, so you should study its output one as many normally functioning systems as you can.

Obtaining Virtual Memory Statistics

The `vmstat` utility is similar to `sar` and `iostat` in broad strokes, but it specializes in reporting on virtual memory (swap space) data. Its syntax is similar to that of `sar`:

```
vmstat [options] [interval [count]]
```

Table 3.6 summarizes some of the more common `vmstat` options. If used without options, `vmstat` produces a CPU report followed by a report on block devices, similar to what typing `vmstat -cd` produces.

TABLE 3.6 Common vmstat Options

Option Name	Meaning
-a	Reports on active and inactive memory use
-f	Displays the number of process forks since the system booted
-s	Displays a variety of memory and virtual memory statistics, including pages swapped in and out, total available memory, and so on
-S [unit]	Changes reporting units to 1000 bytes (k), 1024 bytes (K), 1,000,000 bytes (m), or 1,048,576 bytes (M)

Setting Process Permissions

Most Linux programs run with the permissions of the user who executed them. For instance, if `jane` runs a program, that program can read precisely the same files that `jane` can read. A few programs, though, need additional privileges. For instance, `su`, which allows one user to take on another's identity, requires root privileges to do this identity switching. Such programs use the set user ID (SUID) bit to have the program run with the privileges of the program file's owner. That is, the SUID bit alters the *effective user ID*. The set group ID (SGID) bit works in a similar manner, but it sets the group with which the process is associated. Although these features are useful and even occasionally necessary, they're also at least potential security risks, so you should be sure that as few programs use these features as possible.

Understanding the Risks of SUID and SGID Programs

There are two major potential risks with SUID and SGID programs:

- If the program allows users to do something potentially dangerous, ordinary users might abuse the program. For instance, Linux's `fdisk` program can modify a disk's partitions, potentially leading to a completely destroyed system if abused. Even comparatively innocuous programs like `cp` could be abused if set to be SUID `root`—if so configured, any user could copy any file on the computer, which is clearly undesirable in the case of sensitive files. For these reasons, neither `fdisk` nor `cp` is normally installed as an SUID program.
- Bugs in SUID and SGID programs can cause damage with greater than normal privileges. If some random program contains a bug that causes it to try to recursively remove all files on the computer, and if an ordinary user encounters this bug, Linux's filesystem security features will minimize the damage. If this program were SUID `root`, though, the entire system would be wiped out.

For these reasons, only programs that absolutely require SUID or SGID status should be so configured. Typically, these are programs that ordinary users might reasonably be expected to use and that require privileged access to the system. The programmers who write such programs take great pains to ensure they're bug free. The `root` user may set *any* program's SUID or SGID bit, though.

Knowing When to Use SUID or SGID

SUID and SGID are necessary when a program needs to perform privileged operations but may also legitimately be run by ordinary users. Some common programs that meet this description include `passwd`, `gpasswd`, `crontab`, `su`, `sudo`, `mount`, `umount`, and `ping`. This list is not complete, however.

You can remove the SUID bits on some of these programs, but that may mean that ordinary users won't be able to use them. Sometimes this may be acceptable—for instance, you might not want ordinary users to be able to mount and unmount filesystems. Other times, ordinary users really do need access to these facilities. The `su` utility is the best way for you to acquire root privileges in many cases, for instance; and ordinary users should be able to change their passwords with `passwd`.

Some programs have SUID or SGID bits set, but they aren't SUID or SGID root. These programs may need special privilege to access hardware device files or the like, but they don't need full root privilege to do so. For instance, some older distributions configured their `xterm` programs in this way. Such configurations are much less dangerous than SUID root programs because these special users typically don't have unusual privileges except to a handful of device or configuration files.

You should be aware of the fact that SUID and SGID bits can be set on directories, as well as on ordinary files. Used in this way, the bits have different meanings than they do on program files: an SUID bit on a directory means that only a file's owner may delete files in the directory (as opposed to anybody with write permission to the directory), and an SGID bit means that files created in the directory will have group ownership assigned to the directory's owner (as opposed to ownership by the user who created the file).

Finding SUID or SGID Programs

You can use the `find` command to locate files with their SUID or SGID bits set. Specifically, you need to use the `-perm` parameter to this command and specify the `s` permission code in the user or group. For instance, the following command locates all SUID or SGID files on a computer:

```
# find / -perm +ug+s
```



Real World Scenario

Controlling Daemon Process Permissions

Servers are run in various ways, as described in Chapter 4. Some of these allow you to set the effective user IDs of the server processes. For instance, both `inetd` and `xinetd` allow you to specify the user under whose name the server runs. Sometimes a server needs to run with root permissions, but other times that's not necessary. You should consult a server's documentation to learn what its requirements are.

Some servers let you adjust their process ownership through configuration files. For instance, Apache lets you adjust the username used on most of its processes with the `User` option in its `httpd.conf` file. (In the case of Apache, one process still runs as root, but it spawns children that run with the ownership you specify.)

You may want to run this command and study the results for your system. If you're uncertain about whether a program should have its SUID or SGID bit set, check its man page and try to verify the integrity of its package using RPM, if your system uses RPM. For instance, type **rpm -V *packagename***. This will turn up changes to the permissions of files in *packagename*, including changes to SUID or SGID bits. (Chapter 7, “Managing Packages and System Backups,” describes the **rpm** command in more detail.) Of course, it's conceivable that a program might have had its SUID or SGID bit set inappropriately even in the original package file.

Running Jobs at Specific Times

Some system maintenance tasks should be performed at regular intervals and are highly automated. For instance, the **/tmp** directory (which holds temporary files created by many users) tends to collect useless data files. Linux provides a means of scheduling tasks to run at specified times to handle such issues. This tool is the cron program, which runs what are known as *cron jobs*. A related tool is **at**, which enables you to run a command on a one-time basis at a specified point in the future, as opposed to doing so on a regular basis, as cron does.

Understanding the Role of Cron

Cron is a *daemon*, which means that it runs continuously, looking for events that cause it to spring into action. Unlike many daemons, which are network servers, cron responds to temporal events. Specifically, it “wakes up” once a minute, examines configuration files in the **/var/spool/cron** and **/etc/cron.d** directories and the **/etc/crontab** file, and executes commands specified by these configuration files if the time matches the time listed in the files.

There are two types of cron jobs: *system cron jobs* and *user cron jobs*. System cron jobs are run as **root** and perform system-wide maintenance tasks. By default, most Linux distributions include system cron jobs that clean out old files from **/tmp**, perform *log rotation* (renaming log files and deleting old ones so that they don't grow to fill the disk), and so on. You can add to this repertoire, as described shortly. Ordinary users can create user cron jobs, which might run some user program on a regular basis. You can also create a user cron job as **root**, which might be handy if you need to perform some task at a time not supported by the system cron jobs, which are scheduled rather rigidly.

One of the critical points to remember about cron jobs is that they run unsupervised. Therefore, you shouldn't call any program in a cron job if that program requires user input. For instance, you wouldn't run a text editor in a cron job. You might, however, run a script that automatically manipulates text files, such as log files.

Creating System Cron Jobs

The `/etc/crontab` file controls system cron jobs. This file normally begins with several lines that set environment variables, such as `PATH` and `MAILTO` (the former sets the path, and the latter is the address to which programs' output is mailed). The file then contains several lines that resemble the following:

```
02 4 * * * root run-parts /etc/cron.daily
```

This line begins with five fields that specify the time. The fields are, in order, the minute (0–59), the hour (0–23), the day of the month (1–31), the month (1–12), and the day of the week (0–7; both 0 and 7 correspond to Sunday). For the month and day of the week values, you can use the first three letters of the name rather than a number, if you like.

In all cases, you can specify multiple values in several ways:

- An asterisk (*) matches all possible values.
- A list separated by commas (such as 0,6,12,18) matches any of the specified values.
- Two values separated by a dash (-) indicate a range, inclusive of the end points. For instance, 9–17 in the hour field specifies a time of from 9 a.m. to 5 p.m.
- A slash, when used in conjunction with some other multivalue option, specifies stepped values—a range in which some members are skipped. For instance, */10 in the minute field indicates a job that's run every 10 minutes.

After the first five fields, `/etc/crontab` entries continue with the account name to be used when executing the program (root in the preceding example) and the command to be run (`run-parts /etc/cron.daily` in this example). The default `/etc/crontab` entries generally use `run-parts`, `cronloop`, or a similar utility that runs any executable scripts within a directory. Thus, the preceding example runs all the scripts in `/etc/cron.daily` at 4:02 a.m. every day. Most distributions include monthly, weekly, daily, and hourly system cron jobs, each corresponding to scripts in a directory called `/etc/cron.interval`, where *interval* is a word associated with the run frequency. Others place these scripts in `/etc/cron.d/interval` directories.



The exact times chosen for system cron jobs to execute vary from one distribution to another. Normally, though, daily and longer-interval cron jobs run early in the morning—between midnight and 6 a.m. Check your `/etc/crontab` file to determine when your system cron jobs run.

To create a new system cron job, you may create a script to perform the task you want performed and copy that script to the appropriate `/etc/cron.interval` directory. When the runtime next rolls around, cron will run the script.



Before submitting a script as a cron job, test it thoroughly. This is particularly important if the cron job will run when you're not around. You don't want a bug in your cron job script to cause problems by filling the hard disk with useless files or producing thousands of e-mail messages when you're not present to quickly correct the problem.

If you need to run a cron job at a time or interval that's not supported by the standard `/etc/crontab`, either you can modify that file to change or add the cron job runtime or you can create a user cron job, as described in the next section. If you choose to modify the system cron job facility, model your changes after an existing entry, changing the times and script storage directory as required.



System cron job storage directories should be owned by root, and only root should be able to write to them. If ordinary users can write to a system cron directory, unscrupulous users could write scripts to give themselves superuser privileges and place them in the system cron directory. The next time cron runs those scripts, the users will have full administrative access to the system.

Creating User Cron Jobs

To create a user cron job, you use the `crontab` utility, not to be confused with the `/etc/crontab` configuration file. The syntax for `crontab` is as follows:

```
crontab [-u user] [-l | -e | -r] [file]
```

If given without the `-u user` parameter, `crontab` modifies the cron job associated with the current user. (User cron jobs are often called *crontabs*, but with that word also used in reference to the system-wide configuration file and the utility itself, this usage can be perplexing.) The `crontab` utility can become confused by the use of `su` to change the current user identity, so if you use this command, it's safest to also use `-u user`, even when you are modifying your own cron job.

If you want to work directly on a cron job, use one of the `-l`, `-e`, or `-r` options. The `-l` option causes `crontab` to display the current cron job, `-r` removes the current cron job, and `-e` opens an editor so that you can edit the current cron job. (`Vi` is the default editor, but you can change this by setting the `VISUAL` or `EDITOR` environment variable, as described in Chapter 2.)

Alternatively, you can create a cron job configuration file and pass the filename to `crontab` using the `file` parameter. For instance, typing `crontab -u tbaker my-cron` causes `crontab` to use `my-cron` for `tbaker`'s cron jobs.

Whether you create the cron job and submit it via the `file` parameter or edit it via `-e`, the format of the cron file is similar to that described earlier. You can set environment

variables by using the form *VARIABLE=value*, or you can specify a command preceded by five numbers or wildcards to indicate when the job is to run. In a user cron job, however, you do *not* specify the username used to execute the job, as you do with system cron jobs. That information is derived from the owner of the cron job. Listing 3.3 shows a sample cron job file. This file runs two programs at different intervals: the `fetchmail` program runs every 30 minutes (on the hour and half hour), and `clean-adouble` runs on Mondays at 2:00 a.m. Both programs are specified via complete paths, but you could include a `PATH` environment variable and omit the complete path specifications.

Listing 3.3: A Sample User Cron Job File

```
SHELL=/bin/bash
MAILTO=tbaker
HOME=/home/tbaker
0,30 * * * * /usr/bin/fetchmail -s
0 2 * * mon /usr/local/bin/clean-adouble $HOME
```

User access to cron can be controlled via the `/etc/cron.allow` and `/etc/cron.deny` files. Add a username to `cron.allow` to explicitly allow that user to use cron; add a username to `cron.deny` to explicitly deny that user the ability to use cron. The keyword `ALL` in either file stands for all users, so you can make access to cron very selective by placing `ALL` in `cron.deny` and then adding only trusted users to `cron.allow`.

Using *at*

Sometimes cron is overkill. You might simply want to run a single command at a specific point in the future on a onetime basis, rather than on an ongoing basis. For this task, Linux provides another command: `at`. In ordinary use, this command takes a single option (although options to fine-tune its behavior are also available): a time. This time can take any of several forms:

Time of day You can specify the time of day as *HH:MM*, optionally followed by *AM* or *PM* if you use a 12-hour format. If the specified time has already passed, the operation is scheduled for the next occurrence of that time—that is, for the next day.

noon, midnight, or teatime These three keywords stand for what you’d expect (teatime is 4:00 p.m.).

Day specification To schedule an `at` job more than 24 hours in advance, you must add a day specification after the time of day specification. You can do this in numeric form, using the format *MMDDYY*, *MM/DD/YY*, or *DD.MM.YY*. Alternatively, you can specify the date as *month-name day* or *month-name day year*.

now + count time-units You can specify a time using the keyword `now`, a plus sign (+), and a time period, as in `now + 2 hours` to run a job in two hours.

When you run `at` and give it a time specification, the program responds with its own prompt, `at>`, which you can treat much like your normal Bash or other command

shell prompt. When you’re done typing commands, press Ctrl+D to terminate input. Alternatively, you can pass a file with commands by using the `-f` parameter to `at`, as in `at -f commands.txt noon` to use the contents of `commands.txt` as the commands you want to run at noon.

The `at` command has several support tools. The most important of these is `atd`, the `at` daemon. This program must be running for `at` to do its work. If it’s not, check for its presence using `ps`, as described earlier, in “Examining Process Lists with *ps*.” If it’s not running, look for a SysV startup script and ensure it’s enabled, as described in Chapter 4.

Other `at` support programs include `atq`, which lists pending `at` jobs; `atrm`, which removes an `at` job from the queue; and `batch`, which works much like `at` but executes jobs when the system load level drops below 0.8.

Getting and Setting Kernel Information

The processes described thus far are ordinary *user-mode* processes—that is, those run by users (including `root`). The Linux kernel, however, is a special type of process. The kernel is the lowest-level code that runs as part of Linux. (In fact, technically Linux *is* the kernel; everything else in a Linux OS runs atop the Linux kernel.) The kernel manages memory, is responsible for doling out CPU time, controls access to hardware, and so on.

Because of the importance of the kernel, various utilities give you the ability to query and configure the kernel in various ways. Simply knowing what kernel you’re running is sometimes very important, so you should understand the tools that provide this information. You can also view or modify various kernel control variables. These variables enable you to fine-tune how the kernel works.

Obtaining Kernel Version Information

The `uname` command is the traditional way to learn about your kernel. Table 3.7 summarizes the options available to this command.

TABLE 3.7 Common `uname` Options

Option Name	Option Abbreviation	Meaning
<code>--all</code>	<code>-a</code>	You can obtain a display of all the available information with this option.
<code>--kernel-name</code>	<code>-s</code>	This option prints the kernel name. On a Linux system, the result is <code>Linux</code> .
<code>--nodename</code>	<code>-n</code>	This option displays the computer’s TCP/IP network hostname.

TABLE 3.7 Common uname Options *(continued)*

Option Name	Option Abbreviation	Meaning
<code>--kernel-release</code>	<code>-r</code>	You can learn the kernel version number, such as 2.6.28, with this option. Note that many distributions add additional code numbers to the kernel version number, as in 2.6.18-92.1.22.e15 for the stock kernel provided with CentOS 5.
<code>--kernel-version</code>	<code>-v</code>	This option presents ancillary compilation information, such as the compilation date and perhaps one or two compilation options. (SMP refers to a kernel that supports multiple CPUs or CPU cores, for example.)
<code>--machine</code>	<code>-m</code>	The computer's machine name, in <code>uname</code> parlance, is the CPU architecture for which the kernel was compiled. When run on a 32-bit x86 system, <code>uname</code> is likely to return <code>i686</code> or something similar; on a 64-bit x86-64 system, the result will be <code>x86_64</code> . Note that this refers to the CPU for which the kernel was compiled, not necessarily the CPU on which it's running. It's possible to run a 32-bit kernel on a 64-bit x86-64 CPU, for instance, in which case <code>uname -m</code> will return <code>i686</code> .
<code>--processor</code>	<code>-p</code>	You can obtain the type of CPU hardware in use with this option. Sometimes this will be the same as what <code>-m</code> returns; other times it will be a new code, such as <code>athlon</code> ; and occasionally the result will be unknown.
<code>--hardware-platform</code>	<code>-i</code>	This option produces yet another code related to the CPU type. This one is a more general CPU family code. For x86 systems (including 32-bit kernels run on x86-64 CPUs), it's likely to be <code>i386</code> ; for x86-64 systems in 64-bit mode, it should be <code>x86-64</code> . This option sometimes returns unknown.
<code>--operating-system</code>	<code>-o</code>	This option normally returns GNU/Linux on a Linux system.

The simplest way to use `uname` is generally to pass it the `-a` option:

```
$ uname -a
Linux halrloprillalar.rodsbooks.com 2.6.28 #17 PREEMPT Thu Jan 15 12:39:27 EST
2009 i686 athlon i386 GNU/Linux
```

This option enables you to examine all of the data that can be obtained from `uname`. You can, of course, pass `uname` a different option if you know you want a specific piece of information. Scripts often use `uname` in this way to help them adapt to different systems. For instance, a script might use `uname -s` to determine whether it's running on a Linux or FreeBSD system and then customize its actions accordingly.

A tool that's similar to `uname` in many ways is the `/proc/version` file. You can use `cat` to display this file:

```
$ cat /proc/version
Linux version 2.6.28 (rodsmith@halrloprillalar.rodsbooks.com) (gcc version
4.3.2 20081105 (Red Hat 4.3.2-7) (GCC) ) #17 PREEMPT Thu Jan 15 12:39:27 EST
2009
```

This example is from the same system that produced the preceding `uname -a` output. As you can see, the two tools provide similar, but not quite identical, information. The `/proc/version` file includes the username of the user who compiled the kernel and the name and version of the compiler that did the job, for instance; but `/proc/version` omits some of the detailed hardware information that `uname` provides.

Setting System Control Data

The `/proc` directory is an interesting one if you want to learn about your computer. As just described, `/proc/version` provides information on your kernel. Other files in `/proc` can be equally informative. This directory doesn't actually point to files on your hard disk, but to pseudo-files that the kernel maintains. These files provide information on the kernel and its subsystems. The `/proc/sys` subdirectory is particularly important because it provides information on an assortment of low-level hardware features.

Rather than dig through the files in the `/proc/sys` directory tree, though, you can use a utility known as `sysctl` to do the job. This program enables you to view and set options in this tree using a *key*, which is a named variable, and a *value*, which is the setting associated with the key. Keys come in multiple parts, separated by dots (.) or slashes (/). These parts correspond to the subdirectories and files in the `/proc/sys` directory tree. For instance, the `kernel.ostype` (or `kernel/ostype`) key refers to the `/proc/sys/kernel/ostype` file. This key (or file) normally contains the value `Linux` on a Linux system.

The syntax for `sysctl` is as follows:

```
sysctl [ options ] [ key | key=value | filename ]
```

Table 3.8 summarizes the options available for `sysctl`.

TABLE 3.8 Common `sysctl` Options

Option Name	Meaning
-a	This option displays all the keys and their associated values. It's used without a key or filename.
-A	This option works much like the -a option. The man page specifies a table form.
-n	You can disable the display of key names with this option.
-N	This option disables display of values.
-q	This option is much like -N, but it affects only standard output.
-e	This option disables the display of errors should <code>sysctl</code> encounter an unknown key.
-w	You can change a setting with this option. You must then pass <i>key=value</i> , where <i>key</i> is a key name and <i>value</i> is the value you want to set.
-p	You can change multiple settings at once by specifying them in a file, whose name you must pass to <code>sysctl</code> .

Most of these options refer to read activity. If you pass `-a` as the first parameter, you can view all of the keys, values, or both. If you don't use `-a`, you'll normally pass a key name to see the associated value or use the `-w` or `-p` option to change one or a set of values. For instance, suppose you want to verify and then reset the computer's hostname and reset various options to their startup values. You might type this:

```
# sysctl kernel.hostname
kernel.hostname = diana.luna.edu
# sysctl -w kernel.hostname=artemis.luna.edu
kernel.hostname = artemis.luna.edu
# sysctl -p /etc/sysctl.conf
```

The final command in this example will be followed by lines that echo the keys and values that are being set from the specified file. In fact, the `/etc/sysctl.conf` file is the name of a

file that holds the computer's startup kernel options. You can peruse and edit this file if you want to make permanent changes to your computer's configuration.



Most of the options that can be set via `sysctl` can also be set via more specialized tools. For instance, the `hostname` command (described in Chapter 8, "Configuring Basic Networking") displays or sets the computer's hostname, much as the `kernel.hostname` key in `sysctl` does. The main advantage of `sysctl`, or of the `/etc/sysctl.conf` file, is that it enables you to set a variety of options using one tool or from one location.

Some kernel configuration tasks require a much more intrusive approach than `sysctl` offers. Most notably, it's sometimes necessary to recompile your kernel. In broad strokes, you can do this by downloading the kernel source code from <http://www.kernel.org> or other sites, uncompressing the tarball, configuring the kernel, compiling it, installing it, and rebooting. This task is more tedious than it is difficult, but it does require a good understanding of your hardware and of the features you need. The advantage of compiling a custom kernel is that, if you do it correctly, you'll have a kernel that's fine-tuned for your computer and its needs. You can include precisely the drivers you need, fine-tune multitasking options, and remove features you know you won't use. Web sites such as <http://www.cyberciti.biz/tips/compiling-linux-kernel-26.html> and <http://www.freeos.com/articles/2589/> describe the process in more detail. The CompTIA Linux+ objectives do not explicitly mention kernel compilation.

Editing Files with Vi

Vi was the first full-screen text editor written for Unix. It's designed to be small and simple. Vi is small enough to fit on tiny, floppy-based emergency boot systems. For this reason alone, Vi is worth learning; you may need to use it in an emergency recovery situation. Vi is, however, a bit strange, particularly if you're used to GUI text editors. To use Vi, you should first understand the three modes in which it operates. Once you understand those modes, you can begin learning about the text-editing procedures Vi implements. This section also examines how to save files and exit from Vi.



Most Linux distributions actually ship with a variant of Vi known as Vim, or "Vi Improved." As the name implies, Vim supports more features than the original Vi does. The information presented here applies to both Vi and Vim. Most distributions that ship with Vim support launching it by typing `vi`, as if it were the original Vi.

Using Vi Modes

At any given moment, Vi is running in one of three modes:

Command mode This mode accepts commands, which are usually entered as single letters. For instance, `i` and `a` both enter edit mode, although in somewhat different ways, as described shortly, and `o` opens a line below the current one.

Ex mode To manipulate files (including saving your current file and running outside programs), you use ex mode. You enter ex mode from command mode by typing a colon (`:`), typically directly followed by the name of the ex mode command you want to use. After you run the ex mode command, Vi returns automatically to command mode.

Edit mode You enter text in edit mode. Most keystrokes result in text appearing on the screen. One important exception is the Esc key, which exits from edit mode back to command mode.



If you're not sure what mode Vi is in, press the Esc key. This will return you to command mode, from which you can reenter edit mode, if necessary.



Vi terminology concerning modes is inconsistent. Ex mode is sometimes called *line mode* or *command-line mode*, and edit mode is sometimes referred to as *text mode*. Some sources don't refer to ex mode as a separate mode at all, but instead refer to extended commands or use similar terms.

Editing Text

As a method of learning Vi, consider the task of editing `/boot/grub/menu.lst` to add a new kernel. (Some distributions call this file `/boot/grub/grub.conf`.) Listing 3.4 shows the original `menu.lst` file used in this example. If you want to follow along, enter it using a text editor with which you're already familiar, and save it to a file on your disk.

Listing 3.4: Sample `/boot/grub/menu.lst` File

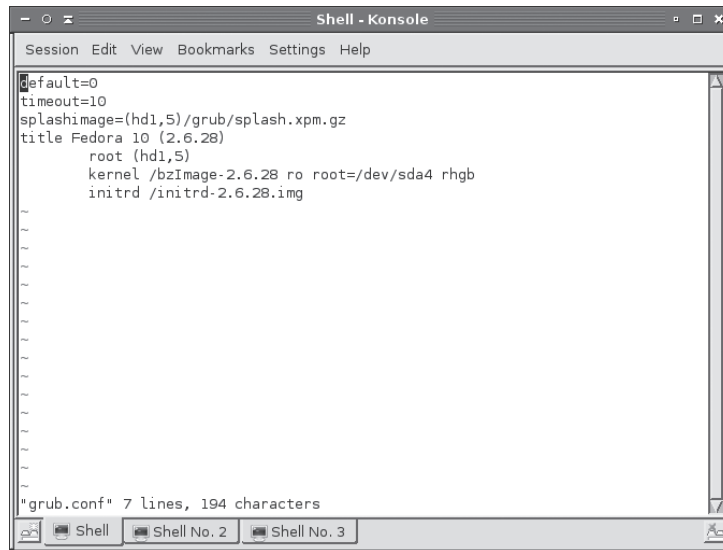
```
default=0
timeout=10
splashimage=(hd1,5)/grub/splash.xpm.gz
title Fedora 10 (2.6.28)
    root (hd1,5)
    kernel /bzImage-2.6.28 ro root=/dev/sda4 rhgb
    initrd /initrd-2.6.28.img
```



Don't try editing your *real* `/boot/grub/menu.lst` file as a learning exercise; a mistake could render your system unbootable! You might put your test `menu.lst` file in your home directory for this exercise.

The first step to using Vi is to launch it and have it load the file. In this example, type `vi menu.lst` while in the directory holding the file. The result should resemble Figure 3.3, which shows Vi running in a Konsole (xterm-like) window. The tildes (~) down the left side of the display indicate the end of the file. The bottom line shows the status of the last command—an implicit file load command because you specified a filename when launching the program.

FIGURE 3.3 Vi's text-mode display provides few “bells and whistles”; the GUI decorations shown here belong to the Konsole window in which Vi is running.



Adding a new entry to `menu.lst` involves duplicating the lines beginning with the `title` line and modifying the duplicates. Therefore, the first editing task is to duplicate these four lines. To do this, follow these steps:

1. Move the cursor to the beginning of the `title` line by using the down arrow key; you should see the cursor resting on the `t`.
2. You must now “yank” four lines of text. This term is used much as “copy” is used in most text editors—you copy the text to a buffer from which you can later paste it back into the file. To yank text, you use the `yy` command, preceded by the number of lines you want to yank. Thus, type **4yy** (*do not* press the Enter key, though). Vi responds with the message `4 lines yanked` on its bottom status line. The `dd` command works much like `yy`, but it deletes the lines as well as copying them to a buffer.

3. Move the cursor to the last line of the file by using the arrow keys. (The cursor may be resting on the line beginning `initrd` or on a line below that, depending on whether the file contains any empty lines.)
4. Type **p** (again, without pressing the Enter key). Vi pastes the contents of the buffer starting on the line after the cursor. The file should now have two identical `title` stanzas. The cursor should be resting at the start of the second one. If you want to paste the text into the document starting on the line *before* the cursor, use an upper-case **P** command.

Now that you've duplicated the necessary lines, you must modify one copy to point to your new kernel. To do so, follow these steps:

1. Move the cursor to the **F** in `Fedora` on the second `title` line. You're about to begin customizing this second stanza.
2. Up until now, you've operated Vi in command mode. You can use several different commands to enter edit mode. At this point, the most appropriate is **R**, which enters edit mode so that it is configured for text replacement rather than insertion. If you prefer insert mode, you could use **i** or **a** (the latter advances the cursor one space, which is sometimes useful at the end of a line). For the purposes of these instructions, type **R** to enter edit mode.
3. Type a new title to replace the current one. For instance, if you're adding a 2.6.29 kernel, you might call this stanza `Fedora 10 (2.6.29)`.
4. Press Esc to return to command mode. You can now use the arrow keys to position the cursor two lines down on the kernel name (`/bzImage-2.6.28` in this example). Type **R** again to enter edit mode. (On some systems, it's possible to reposition the cursor without leaving edit mode, but sometimes attempting to do so causes spurious text entry.)
5. Type a new kernel name, such as `/bzImage-2.6.29`. This label should replace the existing `linux` label.
6. Perform steps similar to the previous two to edit the `initrd` entry for the new kernel's initial RAM disk.
7. Exit from edit mode by pressing the Esc key.
8. Save the file and quit by typing **:wq**. This is actually an `ex` mode command, as described shortly.

Many additional commands are available that you might want to use in some situations. Here are some of the highlights:

Case changes Suppose you need to change the case of a word in a file. Instead of entering edit mode and retyping the word, you can use the tilde (`~`) key in command mode to change the case. Position the cursor on the first character you want to change, and press `~` repeatedly until the task is done.

Undo To undo any change, type **u** in command mode.

Searches To search forward for text in a file, type `/` in command mode, followed immediately by the text you want to locate. Typing `?` will search backward rather than forward.

Global replacement To replace all occurrences of one string by another, type `:%s/original/replacement`, where *original* is the original string and *replacement* is its replacement. Change `%` to a starting line number, comma, and ending line number to perform this change on just a small range of lines.

There's a great deal more depth to Vi than is presented here; the editor is quite capable, and some Linux users are very attached to it. Entire books have been written about Vi. Consult one of these, or a Vi Web page like <http://www.vim.org>, for more information.

Saving Changes

To save changes to a file, type `:w` from command mode. This enters ex mode and runs the `w` ex-mode command, which writes the file using whatever filename you specified when you launched Vi. Related commands enable other functions:

Edit new file The `:e` command edits a new file. For instance, `:e /etc/inittab` loads `/etc/inittab` for editing. Vi won't load a new file unless the existing one has been saved since its last change or unless you follow `:e` with an exclamation mark (!).

Include existing file The `:r` command includes the contents of an old file in an existing one.

Quit Use the `:q` command to quit from the program. As with `:e`, this command won't work unless changes have been saved or you append an exclamation mark to the command.

You can combine ex commands such as these to perform multiple actions in sequence. For instance, typing `:wq` writes changes and then quits from Vi.



Real World Scenario

More Linux Text Editors

For most day-to-day uses, you may prefer to use an editor other than Vi. Emacs is a common choice for this use. Emacs is a very full-featured editor, with plug-ins that handle many traditionally noneditor tasks, such as reading e-mail. Emacs runs both in text mode and in an X window. Like Vi, though, Emacs is a bit strange and intimidating to the uninitiated. Stripped-down text-mode Emacs-like programs, such as `jed` and `nano`, are popular among those who like Emacs but want a slimmer editor. GUI editors, such as `KEdit`, `gedit`, and `NEdit`, are helpful for those who are used to GUI editors in Windows or Mac OS. You can try any of these editors for day-to-day use if you don't like Vi; but be aware that only Vi is mentioned among the Linux+ objectives.

Summary

Many Linux system administration tasks are performed by manipulating files. Linux file-manipulation commands include those for navigating through directories, moving and copying files, manipulating directories, locating files, and examining files. Using redirection and pipes with such commands is also a useful skill to possess. Beyond basic file manipulation commands are commands to actually edit files, such as the Vi editor. Vi is particularly important for system administration because it's a popular editor for inclusion on emergency Linux systems.

Additional system administration tasks involve the manipulation of processes—that is, running programs. You can use tools such as `ps` and `top` to view lists of processes and important process characteristics. Additional tools, such as `nice` and `renice`, enable you to adjust the priorities of processes. You can even terminate errant processes using `kill`. Processes normally run as the user who launched them; however, the SUID and GUID bits on program files enable programs to run as the user or group that owns the file.

The kernel is, in some sense, the master process of a Linux system. Thus, you should be able to learn some details about your kernel and adjust some of its characteristics.

Exam Essentials

Evaluate the need for SUID or SGID programs. Some programs, such as `su` and `passwd`, must have enhanced privileges in order to operate. Most programs, though, do not require these privileges and so should not have their SUID or SGID bits set.

Summarize how to obtain process information. The `ps` command is the premiere tool for producing process listings. Its many options enable you to customize what processes appear in the listing and what information about the processes is displayed. For a dynamically updated process listing, the `top` command does the job.

Understand how to limit the CPU time used by a process. You can launch a program with `nice`, or you can use `renice` to alter its priority in obtaining CPU time. If a process is truly out of control, you can terminate it with the `kill` command.

Know how to create a cron job. You create a system cron job by placing a script in an appropriate directory, such as `/etc/cron.daily`. You can create a user cron job by using the `crontab` command, which enables you to edit a configuration file or pass one to the utility for appropriate handling.

Describe how to identify the kernel a Linux system is running. The `uname` command returns basic kernel and system information, such as the kernel type, version, and target platform. You can obtain specific information using particular options, or you can obtain all the data that `uname` can determine by passing it the `-a` option.

Summarize Vi's three editing modes. You enter text using edit mode, which supports text entry and deletion. The command and ex modes are used to perform more complex commands or run outside programs to operate on the text entered or changed in edit mode.

Review Questions

1. Which of the following commands creates a display of processes, showing the parent/child relationships through links between their names?
 - A. **ps --forest**
 - B. **ps aux**
 - C. **ps -e**
 - D. All of the above
2. What programs might you use to learn what your system's load average is? (Choose all that apply.)
 - A. **ld**
 - B. **load**
 - C. **top**
 - D. **uptime**
3. You use **top** to examine the CPU time being consumed by various processes on your system. You discover that one process, **dfcomp**, is consuming more than 90 percent of your system's CPU time. What can you conclude?
 - A. Very little; **dfcomp** could be legitimately consuming that much CPU time, or it could be an unauthorized or malfunctioning program.
 - B. No program should consume 90 percent of available CPU time; **dfcomp** is clearly malfunctioning and should be terminated.
 - C. This is normal; **dfcomp** is the kernel's main scheduling process, and it consumes any unused CPU time.
 - D. This behavior is normal *if* your CPU is less powerful than a 2.5GHz EM64T Pentium; but on newer systems, no program should consume 90 percent of CPU time.
4. You type **jobs** at a Bash command prompt and receive a new command prompt with no intervening output. What can you conclude?
 - A. The total CPU time used by your processes is negligible (below 0.1).
 - B. No processes are running under your username except the shell you're using.
 - C. The **jobs** shell is installed and working correctly on the system.
 - D. No background processes are running that were launched from the shell you're using.
5. Which two of the following commands are equivalent to one another? (Choose two.)
 - A. **nice --value 10 crunch**
 - B. **nice -n -10 crunch**
 - C. **nice -10 crunch**
 - D. **nice crunch**

6. Which of the following are restrictions on ordinary users' abilities to run `renice`? (Choose all that apply.)
- A. Users may not modify the priorities of processes that are already running.
 - B. Users may not modify the priorities of other users' processes.
 - C. Users may not decrease the priority (that is, increase the priority value) of their own processes.
 - D. Users may not increase the priority (that is, decrease the priority value) of their own processes.
7. You discover that a process with PID 27319 is running out of control, consuming most of the system's CPU time when it shouldn't be. As `root`, you type `kill -15 27319`, but the process continues to run. What might you try next? (Choose all that apply.)
- A. Type `kill -9 27319`.
 - B. Type `kill -TERM 27319`.
 - C. Type `kill -KILL 27319`.
 - D. Type `killall 27319`.
8. Which of the following are risks of SUID and SGID programs? (Choose all that apply.)
- A. The program files are large and thus may cause a disk to run out of space.
 - B. Bugs in the programs may cause more damage than they would in ordinary programs.
 - C. Users may be able to abuse a program's features, thus doing more damage than would otherwise be possible.
 - D. Because the programs require password entry, running them over an insecure network link runs the risk of password interception.
9. Which of the following commands would you type to locate all SUID or SGID programs installed on the computer?
- A. `find / -suid -sgid`
 - B. `find / -perm +u+s`
 - C. `find / -perm +suid+sgid`
 - D. `find / -perm +ug+s`
10. Which of the following lines, if used in a user cron job, will run `/usr/local/bin/cleanup` twice a day?
- A. `15 7,19 * * * tbaker /usr/local/bin/cleanup`
 - B. `15 7,19 * * * /usr/local/bin/cleanup`
 - C. `15 */2 * * * tbaker /usr/local/bin/cleanup`
 - D. `15 */2 * * * /usr/local/bin/cleanup`

11. Which of the following tasks is likely to be handled by a cron job? (Choose all that apply.)
 - A. Starting an important server when the computer boots
 - B. Finding and deleting old temporary files
 - C. Scripting supervised account creation
 - D. Monitoring the status of servers and e-mailing a report to the superuser
12. What is wrong with the following system cron job entry (in `/etc/crontab`)?


```
17 * * * * run-parts /etc/cron.hourly
```

 - A. This command should run hourly but will run only at 5:00 p.m.
 - B. There is no `run-parts` command in Linux.
 - C. The time specification is incomplete.
 - D. It's missing a user specification.
13. A user creates a cron job to retrieve e-mail from a remote server using the `fetchmail` program. What is true of this cron job, if it's properly configured?
 - A. The `fetchmail` process runs with the UID of the user who created the cron job.
 - B. The `fetchmail` process runs with the root UID
 - C. The `fetchmail` process runs with the `crontab` UID.
 - D. The `fetchmail` process runs with the `nobody` UID.
14. At 10:07 a.m., you type **at 9:00**, press the Enter key, and type a command at the `at>` prompt. What will happen, assuming `at` is correctly configured?
 - A. The command you typed will never execute.
 - B. The command you typed will execute at 9:00 a.m. in one year.
 - C. The command you typed will execute at 9:00 p.m.
 - D. The command you typed will execute at 9:00 a.m. the following day.
15. What is the purpose of the `/etc/sysctl.conf` file?
 - A. It holds miscellaneous system configuration options that are set via the `sysctl` utility when the system boots.
 - B. It specifies the order in which system services are started when the computer boots.
 - C. It specifies the filesystems that are mounted at boot time or that may be mounted manually by ordinary users.
 - D. It identifies system services that are started directly by the `init` process when the computer boots.

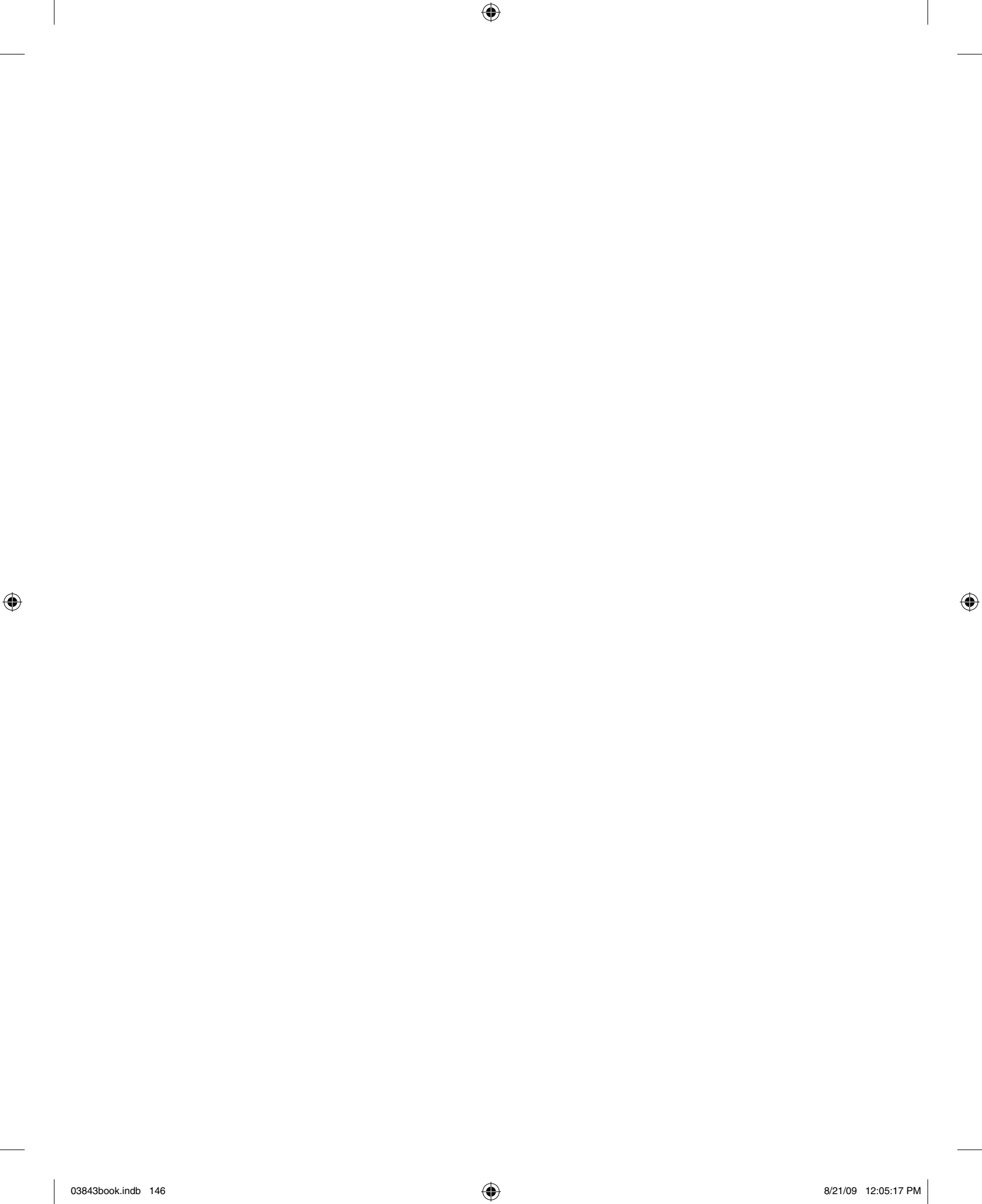
16. You type **uname -a** at a Bash prompt and receive the following output. Which of the following is *not* true of the computer that produced this output?
- ```
Linux aldrin.luna.edu 2.6.28 #6 SMP PREEMPT Wed Jan 14 14:26:19 EST 2009
x86_64 GNU/Linux
```
- A. The kernel supports multiple CPUs (or a multicore CPU).
  - B. The computer's hostname is `aldrin.luna.edu`.
  - C. The computer is running a 32-bit Linux kernel.
  - D. The system is running a 2.6.28 Linux kernel.
17. You edit a file using Vi. When you're done, you hit the Esc key to exit from edit mode, and you type **:q** in command mode to exit from Vi. Vi then complains. Why?
- A. The **:q** command was incomplete; you must type **:q!** to exit from Vi.
  - B. The **:q** command doesn't exit from Vi; it attempts to spell-check the document, but you didn't specify a spell-check dictionary.
  - C. You didn't save your changes; **:q** exits from Vi, but Vi recognizes there are unsaved changes and warns you of this fact.
  - D. You can exit from Vi only using ex mode, which you must enter by typing **:ex** before you type **:q**.
18. You've finished typing a document in Vi, and you want to save it. How would you do this?
- A. Press Esc to enter command mode, and then type **:w** to write the file.
  - B. Select File ➤ Save from the menu.
  - C. Press Ctrl+X and then Ctrl+S.
  - D. Any of the above.
19. Which mode in Vi would you use to type text?
- A. Ex mode
  - B. Command mode
  - C. Type mode
  - D. Edit mode
20. How would you remove two lines of text from a file using Vi?
- A. In command mode, position the cursor on the first line, and type **2dd**.
  - B. In command mode, position the cursor on the last line, and type **2yy**.
  - C. In edit mode, position the cursor at the start of the first line, hold the Shift key down while pressing the down arrow key twice, and hit the Delete key on the keyboard.
  - D. In edit mode, position the cursor at the start of the first line, and press Ctrl+K twice.

## Answers to Review Questions

1. A. The `--forest` option to `ps` shows parent/child relationships by creating visual links between process names in the `ps` output. (Listing 3.1 shows this effect.) Options B and C are both valid `ps` commands, but neither creates the specified effect.
2. C, D. The `top` utility displays a dynamic list of processes ordered according to their CPU use along with additional system information, including load averages. If you want only the load average at a specific moment, `uptime` may be better because it presents less extraneous information—it shows the current time, the time since the system was booted, the number of active users, and the load averages. The `ld` command has nothing to do with displaying load averages (it's a programming tool that links together program modules into an executable program). There is no standard Linux program called `load`.
3. A. CPU-intensive programs routinely consume 90 percent or more of available CPU time, but not all systems run such programs. Furthermore, some types of program bugs can create such CPU loads. Thus, you must investigate the matter more. What is `dfcomp`? Is it designed as a CPU-intensive program? Is it consuming this much CPU time consistently, or was this a brief burst of activity?
4. D. The `jobs` command summarizes processes that were launched from your current shell. When no such processes are running, `jobs` returns nothing, so option D is correct. The `jobs` command doesn't check or summarize CPU load, so option A is incorrect. The `jobs` command also doesn't check for processes run from shells other than the current one, so option B is incorrect (processes running under your username could have been launched from another shell or from a GUI environment). There is no standard `jobs` shell in Linux, so option C is incorrect.
5. C, D. The `nice` command launches a program (`crunch` in this example) with increased or decreased priority. The default priority when none is specified is 10, and the `nice -10 crunch` command also sets the priority to 10, so options C and D are equivalent. Option A isn't a valid `nice` command because `nice` has no `--value` option. Option B is a valid `nice` command, but it sets the priority to `-10` rather than 10.
6. B, D. Linux insulates users' actions from one another, and this rule applies to `renice`; only `root` may modify the priority of other users' processes. Similarly, only `root` may increase the priority of a process, in order to prevent users from setting their processes to maximum priority, thus stealing CPU time from others. Option A correctly describes `nice`, but not `renice`; the whole point of `renice` is to be able to change the priorities of existing processes. Option C also describes an action that `renice` permits.

7. A, C. Signal 15, passed to process 27319 by typing **kill -15 27319**, terminates well-behaved processes but sometimes fails when processes are out of control. Options A and C both pass signal 9 (aka SIGKILL) to the process, which is more likely to work with an out-of-control process but gives the process no chance to shut down cleanly. Another name for signal 15 is SIGTERM, so option B is exactly equivalent to the original command that failed to work. The **killall** command of option D terminates processes by name, so option D will attempt to terminate any process with a *name* (but not a PID) of 27319. Furthermore, **killall** (like **kill**) sends a signal 15 by default, so even if PID 27319 happens to be named 27319, option D won't have any more effect than the original command or option B.
8. B, C. SUID and SGID programs run with effective permissions other than those of the person who runs the program—frequently as **root**. Therefore, bugs or abuses perpetrated by the user may do more damage than could be done if the programs were not SUID or SGID. These programs don't consume more disk space than otherwise identical ordinary programs. Although some SUID and SGID programs ask for passwords (such as **passwd** and **su**), this isn't true of all such programs (such as **mount** and **ping**).
9. D. The **-perm** parameter to **find** locates files with particular permissions, and the **+ug+s** option to **-perm** specifies a permission with the user or group ID bit set; hence, option D will do the job. Option A is incorrect because there are no **-suid** or **-sgid** options to **find**. Option B is almost correct; it will find all files with the SUID bit set, but it won't find files with the SGID bit set (unless the SUID bit is also set). Option C is incorrect because **+suid+sgid** is an invalid option to the **-perm** parameter.
10. B. User cron jobs don't include a username specification (**tbaker** in options A and C). The **\*/2** specification for the hour in options C and D causes the job to execute every other hour; the **7,19** specification in options A and B causes it to execute twice a day, on the 7th and 19th hours (in conjunction with the 15 minute specification, that means at 7:15 a.m. and 7:15 p.m.).
11. B, D. Cron is a good tool for performing tasks that can be done in an unsupervised manner, like deleting old temporary files or checking to see that servers are running correctly. Tasks that require interaction, such as creating accounts, are not good candidates for cron jobs, which must execute unsupervised. Although a cron job could restart a crashed server, it's not normally used to start a server when the system boots; that's done through SysV startup scripts or a super server.
12. D. System cron jobs require a user specification after the time specification and before the command to be executed, and this entry is missing in this specification. (This entry would be legal for a user cron job, though, assuming the user could run the command.) Option A is incorrect because the time specification runs the job at 17 minutes past the hour, every hour; and even if it did run at 5:00 p.m., the entry would be legal, if confusingly named. Option B is incorrect because **run-parts**, although not present on all Linux distributions, is used on several distributions. Cron is also capable of running user-written scripts and programs, so even if **run-parts** weren't a standard Linux utility, the entry would still work if you'd written your own **run-parts** script. Option C is incorrect because the time specification is complete; it includes a minute value (17) and asterisks (\*) denoting a run at every hour, day of the month, month, and day of the week.

13. A. User cron jobs run as the user who created them, so option A is correct.
14. D. The `at` command uses a 24-hour clock, unless you specify `AM` or `PM` suffixes to times, so `at` will interpret `9:00` as being 9:00 a.m. and schedule execution for the next occurrence of this time—that is, 9:00 a.m. the following day, as specified in option D. The `at` utility won't discard the command, schedule execution for a year from now, or schedule execution for 9:00 p.m., as specified by option A, B, or C, respectively.
15. A. Option A correctly describes the purpose of `/etc/sysctl.conf`. Option B is a partial description of the purpose of SysV init scripts. Option C describes the function of the `/etc/fstab` file. Option D describes the purpose of the `/etc/inittab` file.
16. C. The `-a` option to `uname` returns all the information that `uname` can provide. Multiple-CPU support is indicated by the SMP code in the output, the hostname is provided as `aldrin.luna.edu`, and the fact that a Linux kernel with version 2.6.28 is running is evident from those strings in the output. Option C is the correct choice (that is, it's untrue) because the output specifies the machine type as `x86_64`, meaning that the kernel was compiled for an x86-64 CPU, such as an AMD64 or EM64T CPU.
17. C. Vi is smart enough to warn you when you're about to discard changes to your document, such as exiting without saving your changes, as specified in option C. Typing `:q!`, as specified in option A, overrides this check, with the result that your changes will be lost! Option B is fictitious; `:q` has nothing to do with spell-checking. The `:q` command is an ex-mode command, as suggested by option D; however, these commands are entered from command mode by preceding them with colons (`:`).
18. A. Option A correctly describes how to save a file in Vi. (Some other key sequences have similar effects; for instance, you can save and exit from Vi by typing `ZZ` or `:wq` rather than `:w`.) Option B describes the method of saving files in many GUI text editors, but this method doesn't work with Vi. Option C describes the method of saving files in Emacs, but this method doesn't work with Vi.
19. D. Edit mode is used for entering text. Ex mode is used for file operations (including loading, saving, and running external programs). Command mode is used for entering commands of various sorts. There is no "type mode" in Vi.
20. A. In Vi, `dd` is the command-mode command that deletes lines. Preceding this command by a number deletes that number of lines. Although `yy` works similarly, it copies ("yanks") text rather than deleting it. Option C works in many more modern text editors, but not in Vi. Option D works in Emacs and similar text editors, but not in Vi.



# Chapter 4

## Managing System Services

---

### THE FOLLOWING COMPTIA OBJECTIVES ARE COVERED IN THIS CHAPTER:

- ✓ **2.3 Given a scenario, analyze system and application logs to troubleshoot Linux systems (Common log files: /var/log/messages, /var/log/syslog, /var/log/maillog, /var/log/secure, /var/log/lastlog; Rotated logs; Searching and interpreting log files: grep, tail -f, awk, sed).**
- ✓ **2.6 Explain the difference in runlevels and their purpose (Command: init; Runlevels: 0 – Halt, 1 – single-user mode, 2 – single-user mode with networking, 3 – networked multi-user mode, 4 – user configurable, 5 – X11 multi-user mode, 6 – reboot).**
- ✓ **3.1 Manage Linux system services using the following (/etc/init.d: start, stop, restart; inetd; xinetd; chkconfig).**



Chapter 3, “Managing Processes and Editing Files,” described how you can start, stop, and otherwise control ordinary Linux programs. You launch many programs from a Linux shell.

Other programs, though, are launched in other ways. This chapter begins with an examination of this topic, and in particular those programs that are launched automatically when the computer boots. Such processes include server programs, which respond to network access requests, but nonservice programs also run automatically. (Chapter 10, “Configuring Network Servers,” covers server programs in more detail.)

Because system service programs run without user interaction, they use special files, known as *log files*, to record messages for you, the system administrator. This chapter therefore covers Linux’s log files, including both configuring them and using them. A pair of programs that are particularly useful in examining log files, *sed* and *awk*, are also described in this chapter.

## Starting and Stopping Services

Using a command prompt to start every program a Linux system runs is impractical. Programs such as network servers, local login prompts, and the *cron* utility run constantly and should be started automatically with Linux. Such programs are often referred to as *system services*. Several means exist to start and stop such services, including SysV startup scripts, *super servers* (*inetd* or *xinetd*), and local startup scripts. When such a service runs constantly in the background, it’s referred to as a *daemon*. Programs intended to run as daemons often have program names that end in *d*, as in *syslogd* (the system log daemon) or *sshd* (a server for the Secure Shell protocol, SSH). Some servers are themselves run by another server, which is known generically as a *super server*. This practice has certain advantages over running a server directly, as described later in the “Using Super Servers” section.



Linux normally runs any given server using just one of the methods described here, and most distributions provide a single default method of launching a server. This fact is particularly important for SysV startup scripts and *xinetd*, because both of these methods rely on the presence of configuration files that won’t be present if the package maintainer intended that the server be run in some other way.



## Methods of Starting and Stopping Services

Before proceeding further, you should understand the basics of three methods of starting and stopping services: SysV startup scripts, super servers, and local startup scripts. Each approach has its advantages and disadvantages.

SysV startup scripts are named after Unix System V, in which this method first appeared. Each service has a script that controls the service. Depending on the options passed to the script, the script may start the service, shut it down, restart it, or do other things. These scripts are run with specific options when the computer boots or when certain other events occur, as described in the next few sections. SysV startup scripts are typically provided by distribution maintainers with service packages—so a script to launch the Samba server is included with the Samba package. These scripts can launch both network servers and non-server services (such as the `cron` daemon). They're generally used for services that must be running continuously, rather than started on an as-needed basis.

Super servers, by contrast, run continuously, listen for network connections, and launch network servers on an as-needed basis. This approach can reduce the memory load of seldom-used servers, and it also enables the super server, or its helper programs, to perform extra security checks. Super servers impose a speed penalty, though: when a connection request arrives, the super server must launch the program that actually responds to the request, and this takes time, so the computer may take a second or two to respond. Super servers are also useful only for launching network servers, not for non-network services.

Local startup scripts are like SysV startup scripts in that they can launch both network servers and non-network services. These startup scripts are intended to be modified by system administrators to handle local needs, such as unusual configurations or the launching of locally compiled programs for which no SysV startup scripts are available.



### NOTE

The terms *server* and *service* are used in different ways by different people. In this book, I use *server* to refer either to a program that responds to network access requests or to the computer on which a server program runs; I use *service* to refer to any program that runs in the background to handle either network or non-network tasks. Some people, however, restrict the term *server* to computer hardware and use *service* to refer to what I call server programs.

## Starting and Stopping via SysV Scripts

When Linux starts, it enters one of several *runlevels*, each of which corresponds to a specific set of running services, as described in more detail in the upcoming section “Setting the Runlevel.” You can start and stop services controlled through SysV startup scripts either temporarily by running the scripts manually or permanently by setting appropriate links to have the system start or stop the service when it reboots.

**NOTE**

The Gentoo distribution uses named runlevels rather than numbered runlevels. This configuration can be handy if you want to define many runlevels and switch to them by name—say, for using a laptop on any of several networks.

## Temporarily Starting or Stopping a Service

SysV startup scripts reside in particular directories—normally `/etc/rc.d/init.d` or `/etc/init.d`. You may run one of these scripts, followed by an option like `start`, `stop`, or `restart`, to affect the server’s run status. (Some startup scripts support additional options, such as `status`. Type the script name without any parameters to see a list of its options.) For instance, the following command starts the Samba server on an Ubuntu 8.04 system:

```
/etc/init.d/samba start
```

You’ll usually see some indication that the service is starting up. If the script responds with a `FAILED` message, it typically means that something about the configuration is incorrect, or the service may already be running. You should keep a few things in mind when manually starting or stopping a service in this way:

- The name of the startup script is usually related to the package in question, but it’s not fully standardized. For instance, some Samba server packages call their startup scripts `samba`, but others use `smb`. A few startup scripts perform fairly complex operations and start several programs. For instance, many distributions include a `network` or `networking` script that initializes many network functions.
- SysV startup scripts are designed for specific distributions and may not work if you install a package on another distribution. For instance, a Red Hat SysV startup script is unlikely to work properly on a SUSE system.
- A startup script occasionally appears to work when in fact the service doesn’t operate correctly. You can often find clues to failure in the `/var/log/messages` file (type `tail /var/log/messages` to see the last few entries). The upcoming section “Using Log Files” covers these files in more detail.
- One way to reinitialize a server so that it rereads its configuration files is to use the `restart` startup script command. Some server packages provide a `reload` command that makes the server reload its configuration file without shutting down, which is preferable to using `restart` if users are currently using the server. Some startup scripts don’t include a `restart` or `reload` command, though. With these, you may need to manually issue the `stop` command followed by the `start` command when you change configuration options. Some servers provide commands you can issue directly to have them reread their configuration options without explicitly restarting them as well; consult the server’s documentation for details.

Temporarily starting or stopping a service is useful when you need to adjust a configuration or when you first install a server. It's almost always possible to reconfigure a running Linux system without rebooting it by reconfiguring and restarting its services.

## Permanently Starting or Stopping a Service

If you want to permanently change the mix of services your system runs, you may need to adjust which SysV startup scripts the computer runs. As described earlier, Linux determines which services to run by using the runlevel. In addition to the `/etc/rc.d/init.d` or `/etc/init.d` directory in which the SysV startup scripts reside, Linux systems host several directories that contain symbolic links to these scripts. These directories are typically named `/etc/rc.d/rcn.d` or `/etc/rcn.d`, where *n* is a runlevel number. For instance, `/etc/rc.d/rc3.d` is the directory associated with runlevel 3. Gentoo uses named subdirectories of `/etc/runlevels` to define its runlevels—for instance, `/etc/runlevels/default` defines the default runlevel, which the system enters when it boots.

In most distributions, the links in these directories use filenames of the form *Knnservice* or *Snnservice*, where *nn* is a two-digit number and *service* is the name of a service. When the computer enters a given runlevel, it executes the *K\** and *S\** scripts in the associated directory. The system passes the `start` command to the scripts that begin with *S*, and it sends the `stop` command to the scripts that begin with *K*. Thus, the key to controlling the starting and stopping of services is in the naming of the files in these SysV script directories—if you rename a script whose name starts with *S* so that it starts with *K*, it will stop running the next time the system enters the affected runlevel.

The numbers that come after the *S* and *K* codes control the order in which various services are started and stopped. The system executes these scripts from the lowest-numbered to the highest-numbered. This factor can be quite important. For instance, you'll normally want to start network servers like Samba or Apache *after* basic networking is brought up.

Gentoo is an exception to this rule. Its SysV startup script links are not named in any special way. Instead, the Gentoo startup scripts incorporate dependency information, enabling them to start the services on which they rely. This design greatly simplifies SysV administration on Gentoo systems.

Various tools exist to help you adjust what services run in various runlevels. Not all distributions include all these tools, though. The following are some of the tools for adjusting services:

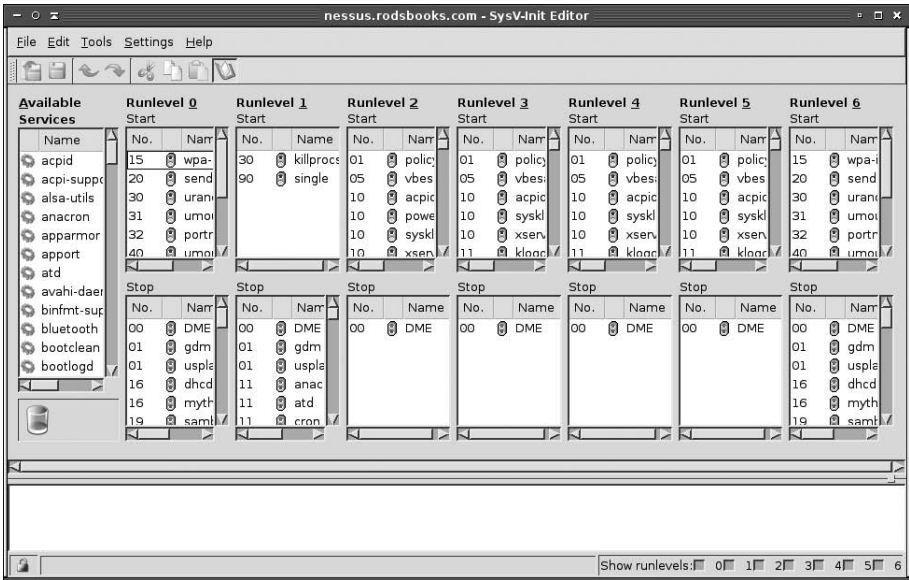
**chkconfig** This command-line utility is most common on Red Hat and related distributions; some don't include it. Pass it the `--list` parameter to see a summary of services and whether they're enabled in each runlevel. You can add or delete a service in a given runlevel by using the `--level` parameter, as in `chkconfig --level 5 smb on`, which enables Samba in runlevel 5. (Pass it off rather than on to disable a service.)

**rc-update** This tool is Gentoo's equivalent of `chkconfig`. To add a script to a runlevel, type `rc-update add script runlevels`, where *script* is the name of the SysV startup script and *runlevels* is one or more runlevel names. Replace `add` with `del` to remove a script from a runlevel. For instance, typing `rc-update add samba default` adds the samba startup script to the default runlevel, causing Samba to run when the system boots.

**ntsysv** This is a text-mode utility that, like `chkconfig`, is most common on Red Hat and related distributions. It presents a menu of services run at the runlevel specified with the `--level` parameter. You can enable or disable a service by moving the cursor to the runlevel and pressing the spacebar.

**ksysv** Figure 4.1 shows this GUI utility. It supports enabling or disabling services in any runlevel from 0 through 6. Locate and select the service in the Start or Stop section of the given runlevel, right-click the entry, and then select Cut from the pop-up menu. This removes its start or stop entry. You can then drag the service from the Available Services list to the runlevel’s Start or Stop list. The system will create an entry in that runlevel and give it a sequence number based on the location to which you dropped it.

**FIGURE 4.1** The `ksysv` program provides a GUI interface to runlevel service management.



**Distribution-specific tools** Many distributions’ general system administration tools, such as Red Hat’s Service Configuration tool and SUSE’s YaST, provide the means to start and stop SysV services in specific runlevels. Details vary from one distribution to another, so consult your distribution’s documentation to learn more.

Once you’ve modified a service’s SysV startup script listings, that service will run (or not run, if you’ve disabled it) the next time you restart the computer or change runlevels, as described in the upcoming section “Setting the Runlevel.” Setting the startup script runlevel information, however, does not immediately run or shut down a service. For that, you’ll need to manually enable or disable the service, as described earlier.



One additional method of permanently disabling a service deserves mention: removing it completely from the computer. You can use a package management system, or you can track down the program's binary files and delete them to ensure that a service never runs. This is certainly the best way to accomplish the task if the computer never needs to run a program, because it saves on disk space and makes it impossible to misconfigure the computer to run an unwanted server—at least, short of reinstalling the server.

## Using Super Servers

Super servers are often used to launch smaller and infrequently used servers. Two super servers are popular on Linux: `inetd` and `xinetd`. You should know both systems, including both basic configurations and security issues related to them.



Be sure you edit the appropriate configuration file! Administrators familiar with one tool are often confused when they work on a system that uses the other super server. The administrator may edit the wrong configuration file and find that changes have no effect. Ideally, a system won't have a configuration file for an uninstalled super server, but sometimes these do exist, particularly when a distribution has been upgraded to a new version that changes the super server.

## Editing *inetd.conf* or *inetd.d* Files

You control servers that launch via `inetd` through the `/etc/inetd.conf` file or files in the `/etc/inetd.d` subdirectory. The `inetd.conf` file consists of a series of lines, one for each server. A typical line resembles the following:

```
ftp stream tcp nowait root /usr/sbin/tcpd /usr/sbin/in.ftpd -l
```



This and several subsequent examples refer to `in.ftpd`, an FTP server that was once quite popular but that's being replaced on many systems by other FTP servers. Some of these servers cannot be run from a super server, so using another server might not work in all of these cases.

Recent versions of `inetd` support splitting the `inetd.conf` file up into several files in `/etc/inetd.d`. This approach enables software packages to include appropriate configuration files, the result being that the server will be set up to work as soon as the package is installed and `inetd` reloaded; there's no need to manually edit `/etc/inetd.conf` after installing a server if its package provides the necessary files. For brevity, the following paragraphs refer to `inetd.conf`, but the format of files in `/etc/inetd.d` is the same.

Each line consists of several fields separated by one or more spaces. Table 4.1 summarizes the meanings of these fields. The basic format is as follows:

*service socket protocol wait/no-wait user server parameters*

**TABLE 4.1** inetd Configuration File Field Meanings

| Field              | Meaning                                                                                                                                                                                                                                                                                                                                                                                                                  |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>service</i>     | The first field (ftp in the preceding example) is the name of the service as it appears in the /etc/services file.                                                                                                                                                                                                                                                                                                       |
| <i>socket</i>      | The socket type entry tells the system what type of connection to expect—a reliable two-way connection (stream), a less reliable connection with less overhead (dgram), a low-level connection to the network (raw), or various others. The differences between these types are highly technical; your main concern in editing this entry should be to correctly type the value specified by the server’s documentation. |
| <i>protocol</i>    | This is the type of TCP/IP protocol used, usually tcp or udp.                                                                                                                                                                                                                                                                                                                                                            |
| <i>wait/nowait</i> | For dgram socket types, this entry specifies whether the server connects to its client and frees the socket (nowait) or processes all its packets and then times out (wait). Servers that use other socket types should specify nowait in this field.                                                                                                                                                                    |
| <i>user</i>        | This is the username used to run the server. The root and nobody users are common choices, but others are possible as well.                                                                                                                                                                                                                                                                                              |
| <i>server</i>      | This is the filename of the server. In the preceding example, the server is specified as /usr/sbin/tcpd, which is the TCP Wrappers binary. This program provides some security checks, enabling you to restrict access to a server based on the origin and other factors. The upcoming section “Controlling Super Server Security” describes TCP Wrappers in more detail.                                                |
| <i>parameters</i>  | Everything after the server name consists of parameters that are passed to the server. If you use TCP Wrappers, you pass the name of the true target server (such as /usr/sbin/in.ftpd) in this field, along with its parameters.                                                                                                                                                                                        |

The hash mark (#) is a comment symbol for /etc/inetd.conf. Therefore, if a server is running via inetd and you want to disable it, you can place a hash mark at the start of the line. If you want to add a server to inetd.conf, you’ll need to create an entry for it. Most servers that can be run from inetd include sample entries in their documentation. Many distributions ship with inetd.conf files that include entries for common servers as well, although many of them are commented out; remove the hash mark at the start of the line to activate the server.

After modifying `inetd.conf`, you must restart the `inetd` super server itself. This super server normally runs as a standard SysV server, so you can restart it by typing something similar to the following:

```
/etc/rc.d/init.d/inetd restart
```

Alternatively, you can tell `inetd` to reload its configuration by passing the SysV startup script the `reload` parameter rather than `restart`. The `restart` option shuts down the server and then starts it again. When you use `reload`, the server never stops running; it just rereads the configuration file and implements any changes. As a practical matter, the two are quite similar. Using `restart` is more likely to correctly implement changes, but it's also more likely to disrupt existing connections.



It's generally wise to disable as many servers as possible in `inetd.conf` (or the `xinetd` configuration files, if you use `xinetd`). As a general rule, if you don't understand what a server does, disable it. This will improve the security of your system by eliminating potentially buggy or misconfigured servers from the equation.

## Editing *xinetd.conf* or *xinetd.d* Files

The `xinetd` (pronounced “zi-net-dee”) program is an extended super server. It provides the functionality of `inetd`, plus security options that are similar to those of TCP Wrappers. Distributions have been slowly shifting from `inetd` to `xinetd`, although some still use `inetd` by default or at least provide it as an option. If you like, you can replace `inetd` with `xinetd` on any distribution.

The `/etc/xinetd.conf` file controls `xinetd`. Typically, though, this file contains only global default options and a directive to include files stored in `/etc/xinetd.d`. Each server that should run via `xinetd` then installs a file in `/etc/xinetd.d` with its own configuration options.

Whether the entry for a server goes in `/etc/xinetd.conf` or a file in `/etc/xinetd.d`, it contains information similar to that in the `inetd.conf` file. The `xinetd` configuration file spreads the information across multiple lines and labels it more explicitly. Listing 4.1 shows an example that's equivalent to the earlier `inetd.conf` entry. This entry provides precisely the same information as the `inetd.conf` entry except that it doesn't include a reference to `/usr/sbin/tcpd`, the TCP Wrappers binary. Because `xinetd` includes similar functionality, it's generally not used with TCP Wrappers.



The upcoming section “Controlling Super Server Security” covers `xinetd` security features.

**Listing 4.1:** Sample xinetd Configuration Entry

```

service ftp
{
 socket_type = stream
 protocol = tcp
 wait = no
 user = root
 server = /usr/sbin/in.ftpd
 server_args = -l
}

```

One additional `xinetd.conf` parameter is important: `disable`. If you include the line `disable = yes` in a service definition, `xinetd` ignores the entry. Many servers install startup files in `/etc/xinetd.d` that have this option set by default; you must edit the file and change the entry to read `disable = no` to enable the server. You can also disable a set of servers by listing their names in the `defaults` section of the main `xinetd.conf` file on a line called `disabled`, as in `disabled = ftp shell`.

As with `inetd`, after you make changes to `xinetd`'s configuration, you must restart the super server. You do this by typing a command similar to the one used to restart `inetd`. As with that command, you can use either `reload` or `restart`, with similar effects. For instance:

```
/etc/rc.d/init.d/xinetd restart
```

## Controlling Super Server Security

Chapter 12, “Securing Linux,” covers many Linux security issues. Because super servers are described here, though, it's appropriate to describe their security features in this chapter as well. You can use a package called TCP Wrappers with either super server to improve security, but it's more commonly used with `inetd`. The `xinetd` super server includes functionality that's similar to TCP Wrappers in its basic feature set.



Whenever possible, apply redundant access controls. For instance, you can use both a firewall and TCP Wrappers or `xinetd` to block unwanted access to particular servers. Doing this helps protect against bugs and misconfiguration—if a problem emerges in the firewall configuration, for instance, the secondary block will probably halt the intruder. If you configure the system carefully, such an access will also leave a log file message that you'll see, so you'll be alerted to the fact that the firewall didn't do its job.

## Controlling Access via TCP Wrappers

The `inetd` super server can be used in conjunction with TCP Wrappers. This package uses a program known as `tcpd`. Instead of having `inetd` call a server directly, `inetd` calls `tcpd`, which does two things: it checks whether a client is authorized to access the server, and if the client has this authorization, `tcpd` calls the server program.



TCP Wrappers is configured through two files: `/etc/hosts.allow` and `/etc/hosts.deny`. The first of these specifies computers that are allowed access to the system in a particular way, the implication being that systems not listed are not allowed access. By contrast, `hosts.deny` lists computers that are not allowed access; all others are given permission to use the system. If a system is listed in both files, `hosts.allow` takes precedence.

Both files use the same basic format. The files consist of lines of the following form:

*daemon-list* : *client-list*

*daemon-list* is a list of servers, using the names for the servers that appear in `/etc/services`. Wildcards are also available, such as `ALL` for all servers.

*client-list* is a list of computers to be granted or denied access to the specified daemons. You can specify computers by name or by IP address, and you can specify a network by using (respectively) a leading or trailing dot (`.`). (Chapter 8, “Configuring Basic Networking,” describes network addressing in more detail.) For instance, `.luna.edu` blocks all computers in the `luna.edu` domain, and `192.168.7.` blocks all computers in the `192.168.7.0/24` network. You can also use wildcards in the *client-list*, such as `ALL` (all computers). `EXCEPT` causes an exception. For instance, when placed in `hosts.deny`, `192.168.7. EXCEPT 192.168.7.105` blocks all computers in the `192.168.7.0/24` network except for `192.168.7.105`.

The `hosts.allow` and `hosts.deny` man pages (they’re actually the same document) provide additional information on more advanced features. You should consult them as you build TCP Wrappers rules.



Remember that not all servers are protected by TCP Wrappers. Normally, only those servers that `inetd` runs via `tcpd` are so protected. Such servers typically include, but are not limited to, Telnet, FTP, TFTP, `rlogin`, `finger`, POP, and IMAP servers. A few servers can independently parse the TCP Wrappers configuration files, though; consult the server’s documentation if in doubt.

## Controlling Access via *xinetd*

Most modern Linux distributions use `xinetd` rather than `inetd`. Although `xinetd` *can* use TCP Wrappers, it normally doesn’t because it incorporates similar functionality of its own. Security is handled on a server-by-server basis through the use of configuration parameters in the `xinetd` configuration files. Some of these parameters are similar to the function of `hosts.allow` and `hosts.deny`:

**Network interface** The `bind` option tells `xinetd` to listen on only one network interface for the server. For instance, you might specify `bind = 192.168.23.7` on a router to have it listen only on the Ethernet card associated with that address. This feature is extremely useful in routers, but it is not as useful in computers with just one network interface. (You can use this option to bind a server only to the loopback interface, `127.0.0.1`, if a server should be available only locally. You might do this with a configuration tool like the Samba Web Administration Tool, or SWAT.) A synonym for this option is `interface`.

**Allowed IP or network addresses** You can use the `only_from` option to specify IP addresses, networks (as in `192.168.78.0/24`), or computer names on this line, separated by spaces. The result is that `xinetd` will accept connections only from these addresses, similar to TCP Wrappers' `hosts.allow` entries.

**Disallowed IP or network addresses** The `no_access` option is the opposite of `only_from`; you list computers or networks here that you want to blacklist. This is similar to the `hosts.deny` file of TCP Wrappers.

**Access times** The `access_times` option sets times during which users may access the server. The time range is specified in the form `hour:min-hour:min`, using a 24-hour clock. Note that this option affects only the times during which the service will *respond*. If the `xinetd` `access_times` option is set to `8:00-17:00` and somebody logs in at 4:59 p.m. (one minute before the end time), that user may continue using the system well beyond the 5:00 p.m. cutoff time.

You should enter these options into the files in `/etc/xinetd.d` that correspond to the servers you want to protect. Place the lines between the opening brace (`{`) and closing brace (`}`) for the service. If you want to restrict *all* your `xinetd`-controlled servers, you can place the entries in the `defaults` section in `/etc/xinetd.conf`.



**NOTE**

Some servers provide access control mechanisms similar to those of TCP Wrappers or `xinetd` by themselves. For instance, Samba provides `hosts.allow` and `hosts.deny` options that work much like the TCP Wrappers file entries, and NIS includes similar configuration options. These options are most common on servers that are awkward or impossible to run via `inetd` or `xinetd`.

## Using Custom Startup Files

Occasionally it's desirable to start a service through some means other than a SysV script or super server. This is most frequently the case when you've compiled a server yourself or installed it from a package file intended for a distribution other than the one you're using, and when you don't want to run it through a super server for performance reasons. In such cases, the program may not come with a SysV startup script, or the provided SysV script may not work correctly on your system.

Many Linux distributions include a startup script that runs after the other SysV startup scripts. This script is generally called `/etc/rc.d/rc.local`, `/etc/rc.d/boot.local`, or something similar. You can launch a server or other program from this script by entering the command you would use to launch the program manually, as described in the program's documentation. For instance, you might include the following line to launch an FTP server:

```
/usr/sbin/in.ftpd -l -D
```

Some programs must have an ampersand (`&`) added to the end of the line to have them execute in the background. If you fail to add this, subsequent lines in the startup script may

not run. Programs intended to run as daemons often don't need the ampersand, but check the documentation to be sure.

One thing to keep in mind when running a server via the local startup script is that this method provides no means to shut down a server, as you can do by passing the `stop` parameter to a SysV startup script. If you want to stop such a server, you'll need to use the Linux `kill` or `killall` command, possibly after locating the server's process ID number via `ps`. For instance, take a look at the following:

```
ps ax | grep ftp
6382 ? S 0:00 in.ftpd -l -D
kill 6382
```



**NOTE**

The `ps` and `kill` commands are covered in more detail in Chapter 3. The `grep` command and the pipe (`|`) are covered in Chapter 2, "Using Text-Mode Commands."

## Setting the Runlevel

One way to change the services a system offers en masse is to change the computer's runlevel. As with individual services, you can change the runlevel either temporarily or permanently. Both can be useful. Temporary changes are useful in testing changes to a system, and permanent changes are useful in creating a system that boots with the desired services running.

### Understanding the Role of the Runlevel

As described earlier in this chapter, Linux enters a specific runlevel when it boots in order to run some predetermined subset of the programs installed on the computer. For instance, you might want to have two configurations for a computer: one that provides all the computer's usual array of network servers and another that provides a more limited set, which you use when performing maintenance on the computer. By defining appropriate runlevels and switching between them, you can easily enable or disable a large number of servers.

On many Linux systems, the runlevel also controls whether the computer provides a GUI or text-mode login prompt. The former is the preferable default state for most workstations, but the latter is better for many server computers or in cases when the X configuration is suspect.

The precise meanings of runlevels vary from one distribution to another. Fedora, Mandriva, Red Hat, SUSE, and several others use the following meanings:

0. Halt the system.
1. Single-user mode.
2. Multiuser mode without networking.
3. Multiuser mode with networking.

4. Unused (may be configured for site-specific purposes).
5. Multiuser mode with networking and an X Display Manager Control Protocol (XDMCP) login.
6. Reboot the system.

Runlevels 0, 1, and 6 have the same meanings across most distributions, but runlevels 2 through 5 have meanings that vary. In particular, Debian and its derivatives, such as Ubuntu, start the system in runlevel 2, which is equivalent to runlevel 3 or 5 in the Red Hat scheme. (X and the XDMCP login are started by SysV startup scripts in these distributions.) Gentoo, as mentioned earlier in this chapter, uses named runlevels.

## Using *init* or *telinit* to Change the Runlevel

The *init* program is critical to Linux's boot process because it reads the `/etc/inittab` file that controls the boot process and implements the settings found in that file. Among other things, *init* sets the system's initial runlevel.

Once the computer has booted, you can use the *telinit* program to alter the runlevel. (In practice, calling *init* directly also usually works.) When using *telinit*, the syntax is as follows:

```
telinit [-t time] runlevel
```



You can discover what runlevel your computer is in with the `runlevel` command. This command displays the previous and current runlevels as output. A previous runlevel of N means that the runlevel hasn't changed since the system booted.

In most cases, `runlevel` is the runlevel to which you want the system to change. There are, however, a few special codes you can pass as well. Most importantly, S or s brings the system into a single-user mode, and Q or q tells the system to reexamine the `/etc/inittab` file and implement any changes in that file.



It's possible to misconfigure X so that it doesn't start. If you do this and your system is set to start X automatically, with some distributions, one consequence is that the system will try to start X, fail, try again, fail, and so on, ad infinitum. If the computer has network connections, one way to stop this cycle is to log in remotely and change the runlevel to one that doesn't start X. This will stop the annoying screen flickering that results as X tries to start and fails. You can then correct the problem from the remote login or from the console, test X, and restore the default runlevel.

When switching runlevels, *init* must sometimes kill processes. It does so “politely” at first by sending a `SIGTERM` signal, which is a way to ask a program to manage its own shutdown. If that doesn't work, though, *init* becomes imperious and sends a `SIGKILL`

signal, which is more likely to work but can be more disruptive because the program may leave temporary files lying about and be unable to save changes to open files. The `-t time` parameter tells `telinit` how long to wait between sending these two signals to a process. The default is 5 seconds, which is normally plenty of time.

One special case of runlevel change happens when you are shutting down the computer. Runlevel 0 shuts down the computer and halts it; depending on kernel options and hardware capabilities, this may shut off power to the computer, or it may simply place the computer in a state from which it's safe to turn off system power. Runlevel 6 reboots the computer. You can enter these runlevels using `telinit`, but it's better to use a separate command called `shutdown` to accomplish this task because it offers additional options. The syntax for this command is as follows:

```
shutdown [-t sec] [-arkhcfF] time [warning-message]
```

Table 4.2 describes the meanings of the parameters.

**TABLE 4.2** shutdown Options and Their Meanings

| Option                 | Meaning                                                                                                                                                                                                                                                                                                                                                               |
|------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| -c                     | If you initiate a shutdown sometime in the future but then change your mind, issuing <code>shutdown</code> again with this parameter cancels it.                                                                                                                                                                                                                      |
| -h                     | This parameter causes the system to halt or power off after a shutdown. Which one occurs is up to the system. (When the system halts, the power stays on, but the OS has terminated. You can typically then power down via a power switch or use the Reset button to reboot the computer.)                                                                            |
| -H                     | This parameter causes the system to halt after a shutdown, but not power off.                                                                                                                                                                                                                                                                                         |
| -k                     | This parameter “fakes” a shutdown—it sends a shutdown warning message to users, but it doesn't shut down the computer.                                                                                                                                                                                                                                                |
| -P                     | This parameter causes the system to power off after a shutdown.                                                                                                                                                                                                                                                                                                       |
| -r                     | This parameter causes a reboot after a shutdown. Essentially, it invokes a change to runlevel 6.                                                                                                                                                                                                                                                                      |
| <i>time</i>            | Shutdowns may be scheduled with this parameter, which can take many different formats. One common value is <code>now</code> , which causes an immediate shutdown. You can also specify a time in 24-hour <i>hh:mm</i> format, as in <code>13:15</code> for a shutdown at 1:15 p.m. A <i>time</i> in the format <code>+m</code> causes a shutdown in <i>m</i> minutes. |
| <i>warning-message</i> | When many people use a system for remote logins, it's generally a good idea to give these users advance warning of a shutdown. You can include a message explaining why the system is going down or how long you expect it to be down.                                                                                                                                |



The parameters to shutdown have changed over the years. Thus, you may find that your version of shutdown has extra parameters or lacks some of those specified in Table 4.2.

On a single-user system, `shutdown -h now` and `shutdown -r now` are perfectly reasonable uses of shutdown. When the system has many users, you might be better off scheduling a shutdown for 5, 10, or more minutes in the future and giving information on the expected downtime, as in the following:

```
shutdown -h +10 "adding new hard disk; up again in 30 minutes"
```



Most modern distributions include commands called `halt` and `reboot` that are equivalent to `shutdown -h now` and `shutdown -r now`, respectively.

## Permanently Changing the Runlevel

You can permanently change the computer's runlevel by editing the `/etc/inittab` file. This file contains a line like the following:

```
id:3:initdefault:
```

This example shows a system configured for runlevel 3. To modify it, you'd change the 3 to whatever value is appropriate. After making this change, you can cause the system to switch immediately to the new runlevel by running `telinit`, as described in the previous section. Typing `telinit Q` will cause the system to read your changes directly, or you can use the runlevel in place of Q.



*Do not* set the default runlevel to 0 or 6 since this will cause the system to shut down or reboot as soon as it boots.



### Real World Scenario

#### Using Upstart

The Linux `init` program is the first process that's launched by the kernel. The traditional version of this program is controlled by `/etc/inittab` and sets the runlevel, launches the login process on several virtual terminals, and starts a few other low-level services. Most Linux systems work fine with a default configuration, with the possible exception of needing to change the runlevel.

Recently, though, the traditional Linux `init` program has been replaced by a new version in some distributions, including Ubuntu and Fedora. This new version, although it still includes `init` and `telinit` commands, goes by a new package name: Upstart (<http://upstart.ubuntu.com>). The Upstart versions of `init` and `inittab` work much like the traditional versions of these tools for most purposes; however, for Upstart, the `/etc/inittab` file is optional. If this file is present, it sets the default runlevel but is otherwise unused. Thus, you can set the default runlevel using `/etc/inittab` on an Upstart-based system, but if you're accustomed to using `/etc/inittab` for other purposes, you'll find that your usual `/etc/inittab` tweaks no longer work.

Instead of using `/etc/inittab` to launch login consoles and other tools, Upstart uses the `/etc/event.d` directory. This directory holds files, each of which describes the conditions for starting and stopping particular `init`-managed services. Chances are you won't need to change these files, but you can do so if you need to alter your virtual terminal configuration, change how the system responds to the `Ctrl+Alt+D` keystroke, or do other things that were handled by `/etc/inittab` in the past.

## Configuring Log Files

Linux maintains log files that record various key details about Linux operation. Using these log files is described later, in “Using Log Files.” You may be able to begin using log files immediately, but knowing how to change the log file configuration can also be important. You do this by configuring the `syslogd` daemon, although some servers and other programs perform their own logging and so must be configured independently. You may even want to configure one computer to send its log files to another system as a security measure. You should also be aware of issues surrounding log file rotation; if your computer doesn't properly manage existing log files, they can grow to consume all available disk space on the partition on which they're stored.

### Understanding *syslogd*

Most Linux systems employ a special daemon to handle log maintenance in a unified way. The traditional Linux system logger is `syslogd`, which is often installed from a package called `sysklogd`. The `syslogd` daemon handles messages from servers and other user-mode programs; it's usually paired with a daemon called `klogd`, which handles kernel messages and is usually installed from the same `sysklogd` package as `syslogd`.



Other choices for system loggers exist. For instance, `syslog-ng` is a replacement that supports advanced filtering options, and `meta-log` is another option. This chapter describes the traditional `syslogd` logger. Others are similar in principle, and even in some specific features, but differ in many details.

The basic idea behind a system logger is to provide a unified means of handling log files. The daemon runs in the background and accepts data delivered from servers and other programs that are configured to use the log daemon. The daemon can then use information provided by the server to classify the message and direct it to an appropriate log file. This configuration enables you to consolidate messages from various servers in a handful of standard log files, which can be much easier to use and manage than potentially dozens of log files from the various services running on the system.

As described in the upcoming section “Using a Remote Server for Log Files,” another feature of log daemons is that they can pass the log information on to a log daemon running on another computer entirely. The advantage of this configuration is that it can help protect the logs from tampering—if a computer is compromised, the intruder can’t eliminate evidence of the intrusion from the log file without first breaking into the computer that logs data for other systems. Administering a network on which all systems log to a single system can also be simpler in some ways, because you can monitor log files on one computer, rather than perform this task on many systems.

In order to work, of course, the log daemon must be configured. In the case of `syslogd`, this is done through the `/etc/syslog.conf` file. The next section describes this file’s format in more detail.

## Setting Logging Options

The format of the `/etc/syslog.conf` file is conceptually simple but provides a great deal of power. Comment lines, as in many Linux configuration files, are denoted by a hash mark (#). Noncomment lines take the following form:

```
facility.priority action
```

In this line, *facility* is a code word for the type of program or tool that has generated the message to be logged; *priority* is a code word for the importance of this message; and *action* is a file, remote computer, or other location that’s to accept the message. The facility and priority are often referred to collectively as the *selector*.

Valid codes for *facility* are `auth`, `authpriv`, `cron`, `daemon`, `kern`, `lpr`, `mail`, `mark`, `news`, `security`, `syslog`, `user`, `uucp`, and `local0` through `local7`. Many of these names refer to specific servers or program classes. For instance, mail servers and other mail-processing tools typically log using the `mail` facility. Most servers that aren’t covered by more specific codes use the `daemon` facility. The `security` facility is identical to `auth`, but `auth` is the preferred name. The `mark` facility is reserved for internal use. An asterisk (\*) refers to all facilities. You can specify multiple facilities in one selector by separating the facilities with commas (,).

Valid codes for *priority* are `debug`, `info`, `notice`, `warning`, `warn`, `error`, `err`, `crit`, `alert`, `emerg`, and `panic`. The `warning` priority is identical to `warn`, `error` is identical to `err`, and `emerg` is identical to `panic`. The `error`, `warn`, and `panic` priority names are deprecated; you should use their equivalents instead. Other than these identical pairs, these priorities represent ascending levels of importance. The `debug` level logs the most information; it’s intended, as the name implies, for debugging programs that are misbehaving. The `emerg` priority logs the most important messages, which indicate very serious problems. When a



program sends a message to the system logger, it includes a priority code; the logger logs the message to a file if you've configured it to log messages of that level or higher. Thus, if you specify a *priority* code of `alert`, the system will log messages that are classified as `alert` or `emerg`, but not messages of `crit` or below. An exception to this rule is if you precede the priority code by an equal sign (=), as in `=crit`, which describes what to do with messages of `crit` priority *only*. An exclamation mark (!) reverses the meaning of a match. For instance, `!crit` causes messages *below* `crit` priority to be logged. A *priority* of `*` refers to all priorities.

You can specify multiple selectors for a single action by separating the selectors by a semicolon (;). Examples appear shortly.

Most commonly, *action* is a filename, typically in the `/var/log` directory tree. Other possibilities include a device filename for a console (such as `/dev/console`) to display data on the screen, a remote machine name preceded by an at-sign (@), and a list of usernames who should see the message if they're logged in. For the last of these options, an asterisk (\*) means all logged-in users.

Some examples should help clarify these rules. First is a fairly ordinary and simple entry:

```
mail.* /var/log/mail
```

This line sends all log entries identified by the originating program as related to mail to the `/var/log/mail` file. Most of the entries in a default `/etc/syslog.conf` file resemble this one. Together, they typically cover all of the facilities mentioned earlier. Some messages may be handled by multiple rules. For instance, another rule might look like this one:

```
*.emerg *
```

This line sends all `emerg`-level messages to the consoles of all users who are logged into the computer using text-mode tools. If this line and the earlier `mail.*` selector are both present, `emerg`-level messages related to mail will be logged to `/var/log/mail` *and* displayed on users' consoles.

A more complex example logs kernel messages in various ways, depending on their priorities:

```
kern.* /var/log/kernel
kern.crit @logger.pangaea.edu
kern.crit /dev/console
kern.info;kern.err /var/log/kernel-info
```

The first of these rules logs all kernel messages to `/var/log/kernel`. The next two lines relate to high-priority (`crit` or higher) messages from the kernel. The first of these lines sends such messages to `logger.pangaea.edu`. (The upcoming section "Using a Remote Server for Log Files" describes remote logging in more detail.) The second of these lines sends a copy of these messages to `/dev/console`, which causes them to be displayed on the computer's main text-mode console display. Finally, the last line sends messages that are between `info` and `err` in priority to `/var/log/kernel-info`. Because `err` is the priority immediately above `crit` and because `info` is the lowest priority, these four lines cause

all kernel messages to be logged two or three times: once to `/var/log/kernel` as well as to either the remote system and the console *or* to `/var/log/kernel-info`.

Most distributions ship with reasonable system logger settings, but you may want to examine these settings and perhaps adjust them. If you change them, though, be aware that you may need to change some other tools. For instance, all major distributions ship with tools that help rotate log files. If you change the files to which `syslogd` logs messages, you may need to change your log file rotation scripts as well.

In addition to the system logger's options, you may be able to set logging options in individual programs. For instance, you might tell programs to record more or less information or to log routine information at varying priorities. Some programs also provide the means to log via the system log daemon or via their own mechanisms. Details vary greatly from one program to another, so you should consult the program's documentation for details.



Most programs that use the system log daemons are servers and other system tools. Programs that individuals run locally seldom log data via the system log daemon, although there are some exceptions to this rule, such as the Fetchmail program for retrieving e-mail from remote servers.

## Rotating Log Files

Log files are intended to retain information on system activities for a reasonable period of time; however, system logging daemons provide no means to control the size of log files. Left unchecked, log files can therefore grow to consume all the available space on the partition on which they reside. To avoid this problem, Linux systems employ *log file rotation* tools. These tools rename and optionally compress the current log files, delete old log files, and force the logging system to begin using new log files.

The most common log rotation tool is a package called `logrotate`. This program is typically called on a regular basis via a cron job. (Cron jobs are described in Chapter 3.) The `logrotate` program consults a configuration file called `/etc/logrotate.conf`, which includes several default settings and typically refers to files in `/etc/logrotate.d` to handle specific log files. A typical `/etc/logrotate.conf` file includes several comment lines, denoted by hash marks (`#`), as well as lines to set various options, as illustrated by Listing 4.2.



Because log file rotation is handled by cron jobs that typically run late at night, it won't happen if a computer is routinely turned off at the end of the day. This practice is common with Windows workstations, but is uncommon with servers. Either Linux workstations should be left running overnight as a general practice or some explicit steps should be taken to ensure that log rotation occurs despite routine shutdowns. You might leave the system up overnight from time to time, for instance, or reschedule the log rotation to some time when the computer is likely to be powered on. The Anacron package (<http://anacron.sourceforge.net>) is another option; it supplements cron by running programs at intervals even when the computer isn't up at specific times.

**Listing 4.2:** Sample `/etc/logrotate.conf` File

```
Rotate logs weekly
weekly

Keep 4 weeks of old logs
rotate 4

Create new log files after rotation
create

Compress old log files
compress

Refer to files for individual packages
include /etc/logrotate.d

Set miscellaneous options
notifempty
nomail
noolddir

Rotate wtmp, which isn't handled by a specific program
/var/log/wtmp {
 monthly
 create 0664 root utmp
 rotate 1
}
```

Most of these lines set options that are fairly self-explanatory or that are well explained by the comments that typically immediately precede them—for instance, the `weekly` line sets the default log rotation interval to once a week. If you see an option in your file that you don't understand, consult the `logrotate` man page.

The last few lines of Listing 4.2 demonstrate the format for the definition for a specific log file. These definitions begin with the filename for the file (multiple filenames may be listed, separated by spaces), followed by an open curly brace (`{`). They end in a close curly brace (`}`). Intervening lines set options that may override the defaults. For instance, the `/var/log/wtmp` definition in Listing 4.2 sets the `monthly` option, which tells the system to rotate this log file once a month, overriding the default `weekly` option. Such definitions are common in the individual files in `/etc/logrotate.d`, which are typically owned by the packages whose log files they rotate. Examples of features that are often set in these definitions include the following:

**Rotated file naming** Ordinarily, rotated log files acquire numbers, such as `messages.1` for the first rotation of the `messages` log file. Using the `dateext` option causes the rotated log

file to obtain a date code instead, as in `messages-20100205` for the rotation performed on February 5, 2010.

**Compression options** As already noted, `compress` causes `logrotate` to compress log files to save space. This is done using `gzip` by default, but you can specify another program with the `compresscmd` keyword, as in `compresscmd bzip2` to use `bzip2`. The `compressoptions` option enables you to pass options to the compression command (say, to improve the compression ratio).

**Creating new log files** The `create` option causes `logrotate` to create a new log file for use by the system logger or program. This option takes a file mode, owner, and group as additional options. Some programs don't work well with the `create` option, though. Most of them use the `copytruncate` option instead, which tells `logrotate` to copy the old log file to a new name and then clear all the data out of the original file.

**Time options** The `daily`, `weekly`, and `monthly` options tell the system to rotate the log files at the specified intervals. These options aren't always used, though; some configurations use a size threshold rather than a time threshold for when to rotate log files.

**Size options** The `size` keyword sets a maximum size for a log file. It takes a size in bytes as an argument (adding `k` or `M` to the size changes it to kilobytes or megabytes). For instance, `size 100k` causes `logrotate` to rotate the file when it reaches 100KB in size.

**Rotation options** The `rotate x` option causes `x` copies of old log files to be maintained. For instance, if you set `rotate 2` for the `/var/log/messages` file, `logrotate` will maintain `/var/log/messages.1` and `/var/log/messages.2`, in addition to the active `/var/log/messages` file. When that file is rotated, `/var/log/messages.2` is deleted, `/var/log/messages.1` is renamed to `/var/log/messages.2`, `/var/log/messages` becomes `/var/log/messages.1`, and a new `/var/log/messages` is created.

**Mail options** If you use `mail address`, `logrotate` will e-mail a log file to the specified address when it's rotated out of existence. Using `nomail` causes the system to not send any e-mail; the log is quietly deleted.

**Scripts** The `prerotate` and `postrotate` keywords both begin a series of lines that are treated as scripts to be run immediately before or after log file rotation, respectively. In both cases, these scripts end with the `endscript` keyword. These commands are frequently used to force `syslogd` or a server to begin using a new log file.

In most cases, servers and other programs that log data either do so via the system logging daemon or ship with a configuration file that goes in `/etc/logrotate.d` to handle the server's log files. These files usually do a reasonable job; however, you might want to double-check them. For instance, you might discover that your system is configured to keep too many or too few old log files for your taste, in which case adjusting the `rotate` option is in order. You should also check the `/var/log` directory and its subdirectories every now and then. If you see huge numbers of files accumulating, or if files are growing to unacceptable sizes, you may want to check the corresponding `logrotate` configuration files. If an appropriate file doesn't exist, create one. Use a working file as a template, modifying it for the new file. Pay particular

attention to the `prerotate` or `postrotate` scripts; you may need to consult the documentation for the program that's creating the log file to learn how to force that program to begin using a new log file.

## Using a Remote Server for Log Files

As noted earlier in “Setting Logging Options,” you can configure `syslogd` to send its logs to a remote computer instead of or in addition to logging data locally. This configuration is fairly straightforward on the system that's doing the logging; in `/etc/syslog.conf`, you provide a computer hostname preceded by an at-sign (`@`) rather than a local filename. For instance, this line causes all kernel messages to be logged to `logger.pangaea.edu`:

```
kern.* @logger.pangaea.edu
```

You can use other selectors, of course, as described earlier in “Setting Logging Options.” Using this feature enables you to search log files for problems from a central location and provides an additional degree of tamper resistance, since an intruder would need to compromise the logging server as well as the primary target of a computer in order to erase evidence of an intrusion from log files.

Ordinarily, `syslogd` 1.3 and later doesn't accept logs sent to it from remote systems. Thus, if you have two computers and configure one computer to send some or all of its logs to the other computer, they won't appear in the logging server's logs by default. To have the logging system accept such submissions, you must launch `syslogd` with its `-r` option. Precisely how you do this varies from one distribution to another. This daemon is normally launched from a SysV startup script, such as `/etc/init.d/syslog`. You may be able to modify this script to pass the `-r` parameter to `syslogd`. Most `syslogd` SysV startup scripts, though, pass parameters to the daemon using a variable, such as `SYSLOGD_PARAMS`. This variable is most frequently set in another file, such as `/etc/sysconfig/rsyslog` (used by Fedora and Red Hat). Some distributions set the variable in the startup script itself; for instance, Debian and Ubuntu set the `SYSLOGD` variable in the `/etc/init.d/syslogd` startup script, enabling you to set this option in the startup script. If you need to change these features, do so and then restart the `syslogd` daemon using its own SysV startup script:

```
/etc/rc.d/init.d/syslog restart
```

The exact name of the SysV startup script varies from one system to another. You must also restart the system logger on the system doing the logging after you make changes to its `/etc/syslog.conf` file. Once this is done, the messages from all the computers configured to log to the logging system should appear in its logs. They should normally be identified by system name:

```
Feb 27 13:17:00 speaker /USR/SBIN/CRON[28223]: (rodsmith) CMD ➤
(/usr/bin/fetchmail -f /home/rodsmith/.fetchmailrc-powweb > /dev/null)
Feb 27 13:18:04 halrloprillalar ntpd[2036]: kernel time sync enabled 0001
```

These lines indicate that the system speaker logged information about a run of `/usr/bin/fetchmail` on February 27 at 13:17:00 (that is, 1:17 p.m.). Soon thereafter, at 13:18:04, the system `hal7lopri11a1ar` recorded activity by the `ntpd` time server.

## Using Log Files

Once you’ve configured logging on your system, this question arises: what can you *do* with log files? Log files are primarily tools in problem solving—debugging servers that don’t behave as you expect, locating evidence of system intrusions, and so on. You should first know what log files to examine in any given situation. Understanding the problem-identification abilities of log files will help you use them effectively. Some tools can help in this task, too; these tools can help you scan log files for information, summarize the (sometimes overly verbose) log file information, and so on.

### Which Log Files Are Important?

In using log files, you must first decide which ones are important. Unfortunately, the names of log files aren’t completely standardized across distributions, so you may need to poke around in your `syslog` configuration files, and perhaps in the log files themselves, to discover which files are important. Table 4.3 summarizes some common log files (all filenames are relative to `/var/log`).

**TABLE 4.3** Common Log Files

| Log File                                    | Purpose                                                                                                                                                                                                                                                                                                             |
|---------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>cron</code>                           | This file holds information on cron-related activity.                                                                                                                                                                                                                                                               |
| <code>dmesg</code> or <code>boot.log</code> | Some distributions place the contents of the kernel ring buffer (described in Chapter 1, “Getting Started with Linux”) in a log file of this name immediately after booting. This ensures that the boot-time kernel ring buffer contents will be accessible even after the system has been running for a long time. |
| <code>lastlog</code>                        | This is an unusual log file; it holds, in binary form, data on the last login times for all users of the system. Type <b>lastlog</b> to see this log file’s data in human-readable form.                                                                                                                            |
| <code>messages</code>                       | This log file sometimes holds general-purpose log data, but on many systems it emphasizes log data from the kernel.                                                                                                                                                                                                 |
| <code>maillog</code>                        | As you might guess by the name, this file holds messages related to the e-mail system.                                                                                                                                                                                                                              |

TABLE 4.3 Common Log Files (continued)

| Log File   | Purpose                                                                                                                                  |
|------------|------------------------------------------------------------------------------------------------------------------------------------------|
| secure     | This log file holds security-related information.                                                                                        |
| syslog     | General-purpose system log data typically ends up in this file.                                                                          |
| Xorg.0.log | This file holds information from the X server about its most recent run. A new log is typically started each time the X server restarts. |

To study your own system’s log files, you should begin by looking over your existing `/etc/syslog.conf` file. Using the information presented earlier in “Setting Logging Options,” you should be able to learn which log files `syslogd` is using on your system, as well as for what purpose these files are being used. This isn’t the end of the story, though; some servers log data without the help of `syslogd`, so you may need to consult the configuration files and documentation for any programs you want to monitor. For instance, Samba frequently logs data independently of `syslogd`, storing files in `/var/log/samba` or a similar directory.

If you’re uncertain of the purpose or importance of a log file, feel free to examine it. The tools described shortly, in “Using Tools to Help Scan Log Files,” can be useful in this task. For basic identification, `less` is likely to be very helpful, as in `less /var/log/messages`. This command displays the file screen by screen, which should give you some clue about the file’s contents.

## Using Log Files to Identify Problems

You can use log files to monitor system loads (for instance, to determine how many pages a Web server has served), to check for intrusion attempts, to verify the correct functioning of a system, and to note errors generated by certain types of programs. To one extent or another, all of these functions can be used to identify problems. Examples of information that can be useful when you are troubleshooting include the following:

**Verifying heavy loads** If a server is running sluggishly, log files may contain clues in the form of a large number of entries from the server. If a server has experienced a massive increase in the number of clients it handles or the size of the files it transfers, you may need to increase the server computer’s capacity to restore good performance. Most nonserver programs don’t log their activities, though, so you probably won’t be able to diagnose similar load problems caused by increasing workstation demands in this way. You’ll likely have an idea that workstation load has increased in a more direct way, though, because the workstation users should know that they’re running more programs or more resource-intensive programs.



Sometimes the logging action itself can contribute substantially to a server’s CPU and disk input/output requirements. If a server is behaving sluggishly, try reducing its logging level (so that it records less information).

**Intrusion detection** Some system problems are related to the presence of an intruder. Interlopers frequently modify your system files or utilities, thus affecting your system's performance or reliability. Their actions are sometimes reflected in log files. Even the *absence* of entries can sometimes be a clue—intruders often delete log files, or at least remove entries for a period. You might not notice such log file discrepancies unless you examine the log files soon after a break-in occurs, however.

**Normal system functioning** If a system is misbehaving, the presence of and information in routine log file entries can sometimes help you pin down the problem, or at least eliminate possibilities. For instance, suppose your system is working as a Dynamic Host Configuration Protocol (DHCP) server for your network, dishing out IP addresses to other systems, as described in Chapter 10. If your clients aren't receiving IP addresses, you can check the log file on the server. If that file indicates that the DHCP server has received requests and given leases in response, you can focus your problem-solving efforts on the clients.

**Missing entries** If you know that a program should be logging information but you can't locate it, this may be evidence that the program is misconfigured or is not starting properly. In some cases, missing entries may indicate problems outside the computer you're examining. For instance, suppose you configure Samba to log access attempts. If you can't access the Samba server from another system, you can check for Samba log file entries. If those entries aren't present, it could mean that Samba isn't running, that it's misconfigured, or that a network problem (such as a misconfigured router or firewall) is blocking access.

**Error messages** The most direct evidence of a problem in a log file is usually an error message. A log file entry that reads `authentication failure` or `FAILED LOGIN` indicates an authentication failure, for instance, that should help you focus your troubleshooting efforts. (Users often see different messages than those that appear in log files, or even none at all.) To improve this capacity, you can configure many servers and utilities to log more information than usual; consult the program's documentation for details. Be aware that different subsystems produce error messages that vary greatly in form, so one program's error messages will look quite different from another's.

Log files are most useful when you are diagnosing software problems with the kernel, servers, user login tools, and miscellaneous other low-level utilities. Information routinely recorded in log files includes kernel startup messages, kernel module operations, user logins, cron actions, filesystem mounting and unmounting, and actions performed by many servers. This information can reflect hardware, kernel, application, configuration, and even user problems.

## Using Tools to Help Scan Log Files

Log files can sometimes be tricky to use because they often accumulate data at a rapid rate. This is particularly true when many programs' logs are sent to a single file or when you've increased the logging level in a program in an effort to help identify problems. Therefore, tools to help scan log files for important information are very helpful. You can think of these tools as falling into one of three categories: those that examine the starts of files, those that examine the ends of files, and those that can be used to search files. Some tools can be used for two or even all three of these tasks.





Most log files are owned by root, and many can be read only by root. Thus, you may need to acquire root privileges before using any of these tools, although the tools themselves can be used by other users on nonlog files.



Most of the commands described here are covered in greater detail in Chapter 2.

## Checking the Beginnings of Log Files

Sometimes you know that information you need appears at the start of a log file. For instance, you might want to study the early stages of the boot process, as recorded in `/var/log/dmesg` or a similar file. You can go about obtaining such information in any of several ways. One tool that's aimed specifically at displaying the beginning of a file is `head`. Used with only a filename as an argument, `head` displays the first ten lines of that file. You can change the number of lines with the `-n` argument, as in `head -n 20 file.txt` to display the first 20 lines of `file.txt`.

If you know the information you want to review is near the beginning of a log file but you're not sure of its exact location, you might prefer to use a pager program, such as `more` or `less`. The `more` program displays a file one screen at a time, whatever your screen size is. You can press the spacebar to move forward in the file a screen at a time. The `less` program's name is a bit of a joke, because `less` is intended to be a better `more`; it does basically the same thing but supports more options within the program, such as searching (described shortly, in "Searching Log Files"). Both programs enable you to quickly check the first few lines of a file, though.

Text editors can also be good ways to check the first few lines in a file. Most text editors open the file and display its first few lines when you pass a filename on the command line. Text editors do have some drawbacks, however. One is that you might accidentally alter the log file, which is undesirable. Another drawback is that opening a log file in a text editor is likely to take longer than using `head` or `less` to display the first few lines. This is particularly true if either the text editor or the log file is unusually large.

## Checking the Ends of Log Files

Information is added to the ends of log files. Thus, when you're performing some operation on a computer and you want to see whether it happened as you intended, that information is likely to appear at the end of a log file, rather than at its start or somewhere in the middle. For instance, when you launch a new server, entries confirming the server's successful startup (or error messages relating to its failure to start) are likely to appear at the end of the file. The ability to check the end of a log file is therefore very helpful.

The `tail` program is noteworthy in this respect because it's designed to display the last few lines (10 by default) of a file. This program is very similar to `head` in most ways, except of course for the fact that it displays the end of a file rather than the beginning. The default

action is sufficient for many purposes if you run the program on a log file immediately after some information has been logged. Sometimes, though, you might need to display a number of lines other than the default of 10. To do this, you use the `-n` option, as in **`tail -n 15 /var/log/messages`** to display the last 15 lines of `/var/log/messages`.

Another feature of `tail` is real-time monitoring—you can use the program to keep an eye on additions to log files as they occur. You might want to do this just before performing some action that you want to monitor; you'll be able to see the relevant log entries as they're added to the log file. To do so, pass the `-f` or `--follow` option to `tail`, as in **`tail -f /var/log/messages`**. The result is an initial display of the last few log entries, as usual; however, `tail` doesn't immediately terminate. Instead, it keeps monitoring the log file and echoes new entries to the screen. When you're done, press `Ctrl+C` to kill `tail` and regain control of your shell.

Although it's not quite as convenient as `tail` for displaying a fixed number of lines, the `less` pager can be useful for checking the end of a log file. Type **`less filename`** to display *filename*; then type **`G`** or press the `Esc` key followed by the greater-than symbol (`>`). This will bring you to the end of the file. If you want to scroll upwards in the file, type **`b`** or press `Esc` followed by `V`. You can scroll back down by typing **`f`**, pressing the spacebar, or pressing `Ctrl+V`. Using these commands, you can quickly examine the final lines of any file, including log files.

As with examining the start of a file, a text editor can be used to examine its end. Load a log file into a text editor, and scroll to the end of the file in whatever way is appropriate. As with examining the start of a file, though, this approach has the drawback that it might result in accidental changes to the file being saved. It might also be slow, particularly on large log files or with large editors. On the other hand, some editors notice when the log file changes and enable you to quickly load the changes. This feature can be handy if you want to monitor changes as they occur.

## Searching Log Files

Sometimes you need to search log files for information. For instance, you might want to see all entries created by Postfix or entries in which you know the string `eth0` appears. You can use any of several text searching tools to help out with such tasks. These tools can search one or more text files and display matching lines, or they can take you to matching lines in these files so that you can examine them in context.

The `grep` command is the most basic of the text-search tools. Type the command, a series of options (including the search string), and a file specification (which typically includes a wildcard) to have it search those files for the specified string. For instance, to find all log entries created by the Postfix mail server, you might type **`grep postfix /var/log/*`**. The result is a series of output lines, each of which begins with the name of the file from which it's taken and concludes with the line in question. (If the string was found in a binary file, `grep` tells you so but doesn't attempt to display the string in context.)

The `grep` command is most useful when searching for entries in multiple log files simultaneously—say, if you don't know to which file a server is logging information. It can also

be useful if you want to display the log entries from a particular server or those that involve a single user or by some other criterion you can easily express as a searchable string.



If you use `grep` to search for a string that's very common, the output is likely to scroll off the top of your screen and possibly exceed the buffer of a scrollable xterm window. This may prevent you from taking a complete census of files in which the string occurs. You can pipe the output through `less`, as in **`grep postfix /var/log/* | less`**, to enable you to scan through the `grep` output in a more controlled way.

Another way to search log files is by using the `less` program. You can use this utility to view a single log file. Once you're viewing a file, press the slash key (/) followed by a search string, as in **`/postfix`** to locate the first occurrence of the string `postfix` in the log file. If that string is present in the file, `less` takes you to that point and highlights the string. Pressing the slash key again moves to the next line that contains the search string. This feature can be handy if you need to see the full context of the line in question. If you want to locate the *last* occurrence of a string, press Esc followed by the greater-than symbol (>) to move to the end of the buffer, and then search backwards using a question mark (?; that is, the slash key with a Shift modifier), as in **`?postfix`**. You can use a text editor to perform similar searches, but with the same caveats described earlier, in “Checking the Beginnings of Log Files”—text editors can be slower than tools such as `less`, and you might accidentally alter the log file.

## Using `sed` and `awk`

Two tools that are often used together are `sed` and `awk`. The command name `sed` stands for “stream editor,” and it's a text editor that uses command-line commands rather than GUI operations or even Vi-style interactive commands to edit a file. The `awk` command name is based on the names of its creators (Alfred J. Aho, Peter J. Weinberger, and Brian W. Kernighan). It's a scripting language that's built around pattern matching. Thus, you can use `awk` to control `sed`, making changes to files based on pattern matches in other files.

Most Linux distributions ship with GNU `awk`, or `gawk`. Although there are a few differences between `gawk` and other `awk` implementations, for the most part they're the same. Most Linux distributions create a symbolic link called `awk` that points to the `gawk` binary. The syntax for `gawk` follows one of two patterns:

```
gawk [options] -f program-file [files]
gawk [options] program-text [files]
```

In the first case, you pass an `awk` program in a separate file (*program-file*); in the second case, you pass the `awk` program (*program-text*) on the same command line as the call to `awk` itself. Typically, this program is enclosed in single quote marks (').

In the context of log file analysis, you might use `grep` to isolate lines of interest in a log file and then use `awk` to isolate the relevant data from the file. For instance, suppose you

want to find the recipients of e-mail processed by your mail server. A typical entry might resemble the following:

```
Apr 2 23:52:45 nessus postfix/smtp[30147]: C4F1E6CDE:↵
to=<jennie@luna.edu>, relay=smtp.abigisp.com[68.1.17.4]:25, delay=0.56,↵
delays=0.02/0.04/0.23/0.27, dsn=2.0.0, status=sent (250 2.0.0 ↵
ars11b0013cx0xk02rs1Gy mail accepted for delivery)
```

The recipient information begins with the string `to=`, so you can use `grep` to locate such lines, eliminating other lines from the input that will be piped into `awk`. You can then use `awk` to isolate the seventh space-delimited field—the one with the recipient information. This is the command to do all of this:

```
$ grep "to=" /var/log/mail.log | awk '/.*/ {print $7}'
```

You might need to adjust this command for your system's log file name and the format of its entries. The output will be a series of lines, such as the following:

```
to=<jennie@luna.edu>,
to=<franklin@example.com>,
```

The `sed` command more directly modifies the contents of files. Its syntax, like `awk`'s, can take one of two forms:

```
sed [options] -f script-file [input-file]
sed [options] script-text [input-file]
```

In either case, *input-file* is the name of the file you want to modify. (Modifications are actually temporary unless you save them in some way.) The script (*script-text* or the contents of *script-file*) is the set of commands you want `sed` to perform. When passing a script directly on the command line, *script-text* is typically enclosed in single quote marks. Table 4.4 summarizes a few `sed` commands that can be used in its scripts.

**TABLE 4.4** Common `sed` Commands

| Command                           | Meaning                                                                                    |
|-----------------------------------|--------------------------------------------------------------------------------------------|
| <code>=</code>                    | Display the current line number                                                            |
| <code>a\text</code>               | Append <i>text</i> to the file                                                             |
| <code>i\text</code>               | Insert <i>text</i> into the file                                                           |
| <code>r filename</code>           | Append text from <i>filename</i> into the file                                             |
| <code>s/regexp/replacement</code> | Replace text that matches the regular expression ( <i>regexp</i> ) with <i>replacement</i> |



Table 4.4 is incomplete; `sed` (like `awk`) is quite complex, and this section merely introduces it.

You can use `sed` to further clean up the output of the preceding `grep/awk` command. For instance, if you want only the e-mail address and not the `to=` string and surrounding punctuation, you could use the following command:

```
$ grep "to=" /var/log/mail.log | awk '/.*/ {print $7}' | sed {s/to=\<///} |>
sed {s/\>\.,//}
```



The backslash character (`\`) is used to signify that the following character should be interpreted as a literal part of the regular expression rather than as a special control character.

The result of the preceding command will be a series of e-mail addresses, one to a line, as in:

```
jennie@luna.edu
franklin@example.com
```

Although they're conceptually simple, both `sed` and `awk` are very complex tools; even a modest summary of their capabilities would fill a chapter. You can consult their man pages for basic information, but to fully understand these tools, you may want to consult a book on the subject, such as Dale Dougherty and Arnold Robbins' *sed & awk, 2nd Edition* (O'Reilly, 1997).

## Using Additional Log File Analysis Tools

Manually examining log files with `tail`, `less`, and similar tools can be informative, but other tools exist to help you analyze your log files. One of these is Logcheck, which is part of the Sentry Tools package (<http://sourceforge.net/projects/sentrytools/>). This package comes with some distributions, such as Debian. Unfortunately, it requires a fair amount of customization for your own system, so it's most easily implemented if it comes with your distribution, preconfigured for its log file format. If you want to use it on another distribution, you must edit the `logcheck.sh` file that's at the heart of the package. This file calls the `logtail` utility that checks log file contents, so you must configure the script to check the log files you want monitored. You can also adjust features such as the user who's to receive violation reports and the locations of files that contain strings for which the utility should look in log files. Once it's configured, you call `logcheck.sh` in a cron job. Logcheck then e-mails a report concerning any suspicious system logs to the user defined in `logcheck.sh` (root, by default).

## Summary

System services are those programs that run behind the scenes on any Linux system. Many system services are network server programs, but others are purely local in action. Knowing how to start and stop system services, both on a temporary and a permanent basis, will give you control over how your computer behaves—what network servers it provides to the world, what local utilities it runs, and so on. Most system services are run via SysV startup scripts or via a super server, but you may also use local startup scripts to do the job.

Because system services don't attach themselves to a console, they don't normally display messages for you on your screen. Instead, they write messages to log files. These messages can include warnings about security violations, complaints about unexpected events, and more routine data about normal operations. Knowing how to examine and search log files will enable you to keep an eye on your system's operation, both to verify normal functioning and to investigate problems—with any luck before they become serious!

## Exam Essentials

**Describe the SysV startup procedure.** The `init` process reads the `/etc/inittab` file or the contents of the `/etc/event.d` directory, which controls programs that run when changing from one runlevel to another. Scripts in directories corresponding to each runlevel (typically `/etc/rc#.d` or `/etc/rc.d/rc#.d`, where `#` is the runlevel number) start and stop services when the runlevel changes.

**Explain the differences between SysV startup scripts and super servers for running servers.** SysV startup scripts start servers running on a computer at startup or when changing runlevels so that the servers are always running and can respond quickly to requests, but servers run in this way consume RAM at all times. Super servers run the target servers only in response to requests from clients, thus reducing the memory burden for infrequently used servers but at the cost of slower responses to incoming requests.

**Summarize how TCP Wrappers or `xinetd` can improve security.** These programs both perform initial security checks on a connection, such as verifying that the IP address of a client system is authorized to use the server. Only when a client passes any initial checks is the server program launched and the connection handed off to the server program.

**Describe the function of the runlevel.** Sometimes you may want to run a Linux system with a different set of services than you run at other times. The runlevel lets you define several sets of services and switch quickly between them.

**Describe the function of a system logger.** A system logger is a daemon that accepts information from servers and other programs that want to record information about their normal operation in standardized log files. The system logger creates and manages these files on behalf of the programs that generate the log entries.

**Explain why using a remote system logger can be beneficial.** A remote system logger is a computer that accepts log entries for other systems. This practice improves overall network security because it protects logs from tampering by intruders—to change a log file, the intruder must compromise two computers rather than one. You can also search consolidated log files much more easily than you can search them on multiple computers.

**Summarize how `tail` and `less` differ as tools for examining log files.** The `tail` command displays the final few lines of a file, which is handy if you know an entry you want to see is at the very end of a log file. The `less` command enables you to page through a file, search its contents, and so on. It's not as convenient as `tail` if you just want to see the last few lines of a file, but it's superior if you need to search for information or aren't sure precisely how many lines you need to examine.

## Review Questions

1. You've located a program that continuously monitors your disk activity in the background and e-mails you if it detects problems. Which of the following methods is *least* appropriate for launching this program?
  - A. Typing its name at a Bash prompt
  - B. A SysV startup script
  - C. A super server
  - D. A local startup script
2. A Linux system keeps its SysV startup scripts in the `/etc/init.d` directory. Which of the following commands will temporarily stop the ProFTPD server on that computer, if it's started from these startup scripts?
  - A. `/etc/init.d/proftpd stop`
  - B. `sysvstop /etc/init.d/proftpd`
  - C. `sysvstop proftpd`
  - D. `/etc/init.d/proftpd stop5m`
3. You've installed a server by compiling it from source code. The source code included no SysV startup script, and you don't want to run it from a super server, so you start it in a local startup script (`/etc/rc.d/rc.local`). You need to temporarily shut down the server. How might you do this?
  - A. Type `/etc/rc.d/rc.local stop`.
  - B. Edit the startup script to remove the server, and rerun the script.
  - C. Remove the server's entry from `/etc/inetd.conf`, and type `/etc/rc.d/init.d/inetd restart`.
  - D. Find the server's process ID number (*pid*) with `ps`, and then type `kill pid`.
4. You've discovered that the VitalServer package is installed on your Fedora system, but it's not running automatically when you start the computer into runlevel 5. Assuming this package comes with a SysV startup script called `vitald`, what might you type to enable this server in runlevel 5?
  - A. `xinetd 5 vitald add`
  - B. `init vitald 5`
  - C. `rc-update add vitald 5`
  - D. `chkconfig --level 5 vitald on`



5. Why might you use `reload` rather than `restart` as a parameter to a SysV startup script after making changes to a server's configuration file?
  - A. The `reload` option reloads an individual server, whereas the `restart` option causes Linux to shut down and reboot.
  - B. For some servers, `reload` causes the server to reload the configuration file without shutting down, whereas `restart` fully shuts down and restarts the server, which can disconnect users.
  - C. The `reload` option rereads the configuration file using existing file handles, whereas `restart` creates new file handles, increasing the risk of file corruption on the disk.
  - D. For some servers, `reload` works more reliably because it performs a more complete set of changes to the configuration than does `restart`.
6. A new Linux system administrator edits `/etc/inetd.conf` to add a server. After making this change, the administrator tests the new server, but a remote system can't access the new server. Why might this be? (Choose all that apply.)
  - A. The administrator may have forgotten to restart `inetd`.
  - B. The system might be using `xinetd` rather than `inetd`.
  - C. The administrator may have forgotten to edit the `/etc/rc.d/init.d` script for the new server.
  - D. The administrator may have forgotten to start the new server manually for the first time.
7. A server/computer combination appears in both `hosts.allow` and `hosts.deny`. What's the result of this configuration when TCP Wrappers runs?
  - A. TCP Wrappers refuses to run and logs an error in `/var/log/messages`.
  - B. The system's administrator is paged to decide whether to allow access.
  - C. `hosts.deny` takes precedence; the client is denied access to the server.
  - D. `hosts.allow` takes precedence; the client is granted access to the server.
8. When is the `bind` option of `xinetd` most useful?
  - A. When you want to run two servers on one port
  - B. When you want to specify computers by name rather than IP address
  - C. When `xinetd` is running on a system with two network interfaces
  - D. When resolving conflicts between different servers
9. To alter a Linux system's default runlevel, what would you do?
  - A. Issue the `telinit x` command, where `x` is the desired runlevel.
  - B. Edit `/etc/modules.conf`, and enter the runlevel as an option to the `runlevel` module.
  - C. Issue the `telinit Q` command to have the system query you for a new runlevel.
  - D. Edit `/etc/inittab` and enter the correct runlevel in the `initdefault` line.

10. Which of the following commands switches a running system into runlevel 3?
- A. `telnet 3`
  - B. `runlevel 3`
  - C. `telinit 3`
  - D. `switch-runlevel 3`
11. What does the following command, when typed by a system administrator at noon, accomplish?
- ```
# shutdown -r 01:00 "Up again soon."
```
- A. Reboots the computer at 1:00 p.m. (in one hour) and displays the message `Up again soon` as a warning to users
 - B. Shuts down (halts) the computer at 1:00 p.m. (in one hour) and displays the message `Up again soon` as a warning to users
 - C. Shuts down (halts) the computer at 1:00 a.m. (in 13 hours) and displays the message `Up again soon` as a warning to users
 - D. Reboots the computer at 1:00 a.m. (in 13 hours) and displays the message `Up again soon` as a warning to users
12. What is the difference between runlevels 3 and 5 on a standard Red Hat system?
- A. Runlevel 3 doesn't start X, but runlevel 5 does.
 - B. Runlevel 3 doesn't launch SysV init scripts, but runlevel 5 does.
 - C. Runlevel 3 launches into single-user mode, whereas runlevel 5 is multiuser.
 - D. All of the above.
13. You want to shut down X on an Ubuntu system, so you enter runlevel 3; however, X doesn't shut down. Why not?
- A. Your X server is misconfigured to run in all runlevels.
 - B. Ubuntu doesn't use runlevels to control when X runs.
 - C. You must also type **stopx** to stop X.
 - D. You must enter runlevel 4 on Ubuntu to shut down X.
14. Which of the following system logging codes represents the *highest* priority?
- A. `emerg`
 - B. `warning`
 - C. `crit`
 - D. `debug`

15. Which of the following is an advantage of designating one well-protected computer to record log files for several other computers?
 - A. Logging information in this way minimizes network use.
 - B. The logging system can analyze the logs using Tripwire.
 - C. Logs stored on a separate computer are less likely to be compromised by an intruder.
 - D. You can log information to a separate computer that you can't log locally.
16. Why is a log file analysis tool like Logcheck useful?
 - A. Logcheck translates log file entries from cryptic comments into plain English.
 - B. Logcheck sifts through large log files and alerts you to the most suspicious entries.
 - C. Logcheck compares patterns of activity across several days or weeks and spots anomalies.
 - D. Logcheck uses information in log files to help identify a cracker.
17. Which of the following configuration files does the `logrotate` program consult for its settings?
 - A. `/etc/logrotate.conf`
 - B. `/usr/sbin/logrotate/logrotate.conf`
 - C. `/usr/src/logrotate/logrotate.conf`
 - D. `/etc/logrotate/.conf`
18. Your manager has asked that you configure `logrotate` to run on a regular, unattended basis. What utility/feature should you configure to make this possible?
 - A. `at`
 - B. `logrotate.d`
 - C. `cron`
 - D. `inittab`
19. Which of the following commands will change all occurrences of `dog` in the file `animals.txt` to `mutt` in the screen display?
 - A. `sed -s "dog" "mutt" animals.txt`
 - B. `grep -s "dog||mutt" animals.txt`
 - C. `sed 's/dog/mutt/' animals.txt`
 - D. `cat animals.txt | grep -c "dog" "mutt"`
20. Which of the following is an advantage of `xinetd` over `inetd`?
 - A. `xinetd` can control SysV startup scripts; `inetd` can't do this.
 - B. `xinetd` includes more built-in security features than does `inetd`.
 - C. `xinetd` provides a more succinct one-line-per-server configuration file format.
 - D. `xinetd` launches X; systems using `inetd` must launch X separately.

Answers to Review Questions

1. C. Super servers listen for network connections and launch network servers in response to connection attempts. They are useful for launching other servers, but the described program isn't a server and therefore would not normally be launched in this way, so option C is correct. Option A is a reasonable way to launch the program for testing or occasional use, although it would be awkward if you want to always use the program. Options B and D are both perfectly reasonable ways to launch the described program on a regular and automatic basis.
2. A. There is no standard `sysvstop` command, so options B and C can't be correct. Option D uses a parameter (`stop5m`) that's not standard, and so it won't stop the server. Option A stops the server, which can be manually restarted later or which will restart automatically when the system is rebooted, if it's configured to do so.
3. D. Killing the server with `kill` will stop a running server. Local startup scripts don't accept `start` and `stop` parameters like those used by SysV startup scripts. Rerunning the startup script, even after editing it to remove references to the target server, won't kill running processes. `inetd` is a super server, and since the server in question isn't being run from a super server, restarting `inetd` won't kill the target server.
4. D. The `chkconfig` utility, present in Red Hat, Fedora, and several other distributions, may be used to modify what services are launched by the SysV startup scripts. Option D presents the correct syntax for starting the `vitald` service in runlevel 5. Option A is incorrect because `xinetd` is a super server; it doesn't manage SysV startup scripts. Option B is incorrect because the `init` program is the first program run by the kernel. Although it may be called subsequently to change runlevels, `init` doesn't start or stop individual SysV services. Option C is incorrect because `rc-update` is a Gentoo-specific tool for managing system services, but the question specified a Fedora system. (On a Gentoo system, typing **`rc-update add vitald default`** would have the desired effect.)
5. B. Option B correctly summarizes the difference between these two options, when both are implemented. (Some SysV startup scripts lack a `reload` option.) Option A is incorrect because no standard SysV startup script causes the system to restart. Option C is entirely fictitious. Option D is, if anything, backwards; `restart` is more likely to work than `reload` for some servers.
6. A, B. After editing `/etc/inetd.conf`, `inetd` should be restarted, typically by typing **`/etc/rc.d/init.d/inetd restart`** or something similar. An unused `/etc/inetd.conf` file can sometimes lure administrators used to configuring this file into editing it rather than configuring `xinetd` on systems that run this alternative super server. Running or editing the target server's startup script is unnecessary in this scenario because the server is started from the super server; it's not run directly.
7. D. TCP Wrappers favors `hosts.allow` when a server/computer combination appears in both `hosts.allow` and `hosts.deny`. TCP Wrappers uses this feature to allow you to override broad denials by adding more specific explicit access permissions to `hosts.allow`, as when setting a default deny policy (`ALL : ALL`) in `hosts.deny`. Options A and B are both simply wrong; TCP Wrappers does neither of these things. Option C is backwards.

8. C. The `bind` option of `xinetd` lets you tie a server to just one network interface, rather than link to them all. It has nothing to do with running multiple servers on one port, specifying computers by hostname, or resolving conflicts between servers.
9. D. The `/etc/inittab` file controls the default runlevel. Although `telinit` can be used to *temporarily* change the runlevel, this change will not be permanent. The command `telinit Q` tells the system to reread `/etc/inittab`, so it could be used to implement a changed default after you've edited the file, but it will have no effect before editing this file. The `/etc/modules.conf` file has nothing to do with runlevels, and there is no standard `runlevel` module.
10. C. The `telinit` command changes runlevels. Option A, `telnet`, is Linux's Telnet client for initiating remote logins. Option B, `runlevel`, displays the current and previous runlevel but doesn't change the runlevel. There is no `switch-runlevel` command (option D).
11. D. The reboot time, when specified in `hh:mm` form, is given as a 24-hour clock time, so 01:00 corresponds to 1:00 a.m. The `-r` parameter specifies a reboot, not a halt. (`-h` specifies a halt.)
12. A. Option A correctly describes the meanings of these two runlevels. Contrary to option B, SysV init scripts are used in all runlevels on Red Hat (and most other Linux systems). Option C describes the difference between runlevels 1 and 2–5, not between 3 and 5.
13. B. Ubuntu runs X in all its multiuser runlevels by default, so entering runlevel 3 will have no effect on X; you must use a SysV startup script to shut down X on Ubuntu. Option A is incorrect because Ubuntu's standard configuration isn't a misconfiguration, although it is different from Red Hat's system; and it's not the X server that controls what runlevels it's run in, but your SysV and runlevel configuration. Option C is incorrect because there's no `stopx` command, although there is a `startx` command that launches X from a text-mode login. Contrary to option B, X won't shut down in runlevel 4 in a standard Ubuntu configuration.
14. A. The `emerg` priority code is the highest code available and so is higher than all the other options. (The `panic` code is equivalent to `emerg` but isn't one of the options.) From highest to lowest priorities, the codes given as options are `emerg`, `crit`, `warning`, and `debug`.
15. C. Intruders often try to doctor system logs to hide their presence. Placing logs on another computer makes it less likely that they'll be able to achieve this goal, so you're more likely to detect the intrusion. Logging to a separate computer actually *increases* network use. Tripwire doesn't do log analyses; that job is done by Logcheck, and Logcheck can run on any computer that stores logs. System loggers can record any information locally that can be logged remotely.
16. B. Logcheck uses pattern-matching rules to extract log file entries containing keywords associated with suspicious activity. Although the other options might be useful to have, Logcheck and other common log file analysis tools cannot perform these tasks.
17. A. The `logrotate` program consults a configuration file called `/etc/logrotate.conf`, which includes several default settings and typically refers to files in `/etc/logrotate.d` to handle specific log files.

18. C. The `logrotate` program can be started automatically—and unattended—on a regular basis by adding an entry for it in `cron`. The `at` utility would be used if you wanted the program to run only once, while `logrotate.d` defines how the program is to handle specific log files. The `init` table is used for services and startup and not for individual programs.
19. B. The `sed` utility can be used to “stream” text and change one value to another. In this case, the `s` option is used to replace `dog` with `mutt`. The syntax in option A is incorrect, while options B and D are incorrect since `grep` does not include the functionality needed to make the changes.
20. C. `xinetd` includes security features that enable you to control what remote systems can connect to specific `xinetd`-managed servers. If you use `inetd`, you must add the separate TCP Wrappers package to obtain similar functionality. Thus, option B is correct. Option A is incorrect because neither `inetd` nor `xinetd` is used to control SysV startup scripts (although both programs are typically launched from SysV scripts). Option C is incorrect because `inetd` uses a one-line-per-server configuration file format; `xinetd` uses a more verbose multi-line format. Which is better is a matter of personal preference. Option D is incorrect because neither `inetd` nor `xinetd` launches X. (Either can launch certain X-related tools, though, such as an XDMCP login server.)

Chapter 5

Managing Users

THE FOLLOWING COMPTIA OBJECTIVES ARE COVERED IN THIS CHAPTER:

- ✓ **5.1 Manage and monitor user and group accounts using the following (Tools: useradd, userdel, usermod, groupadd, groupdel, groupmod, **lock**, who, w, last, whoami; Files: /etc/skel, /etc/passwd, /etc/shadow, /etc/group).**
- ✓ **5.4 Given a scenario, implement privilege escalation using the following (sudo, su).**



Traditional PC OSs, such as DOS and early versions of Windows, are basically single-user OSs. Although it's certainly possible for two or more people to use computers running these OSs, the OSs themselves provide no mechanisms to help keep users from reading or even damaging one another's files. Linux, on the other hand, is modeled after Unix, which was designed as a multiuser OS. In Linux and Unix, the OS provides tools designed to help keep users from harming one another's files. The same mechanisms are used to provide security and to keep users from damaging the OS as a whole. For these reasons, Linux system administrators must understand how the OS handles users and what tools are available to help you manage the users on your own system.

This chapter covers several specific user management topics, starting with an overview of basic multiuser concepts. Next up is information on configuring users and groups of users, as well as common strategies you can employ in managing users and groups. Because Linux's account system is a pillar in its security system, this chapter describes policies you can use in account management to improve security, focusing on good password practices. This chapter concludes with a look at access control—using accounts, encryption, and server-specific options to limit access to the computer by particular users or computers.

Understanding Multiuser Concepts

Before dealing with the nitty-gritty details of administering user accounts on a Linux system, you should understand the underlying concepts, including a few implementation details. Knowing this information will help you plan an effective account structure or expand an existing one to meet new needs. This information may also be critically important when you're moving accounts from one computer to another, adding a new hard disk, or performing other types of system maintenance.

User Accounts: The Core of a Multiuser System

Linux user accounts are basically the same as user accounts in other Unix-like OSs. They enable several people to use the same system, either at different times or at the same time, without interfering with one another. A single user can even have several simultaneous logins active, which is sometimes convenient. It's important to understand what user accounts allow you to do with a system and also how users are identified.

Accounts in a Multiuser System

Technically, a user is a person, whereas an account is a set of data structures and permissions associated with that user. Frequently, though, the term *user* is used as if it were synonymous with *account*, as in “you must delete this user.” Don’t take such language literally—delete the account, not the user.

Several important features have been associated with Linux accounts, including the following:

Username The *username* is the name by which the account is known to humans, such as `ellen`. The characteristics of Linux usernames are described in more detail shortly, in “Linux Usernames.”

Login privileges An account enables an individual to log into a Linux computer. Depending on the system’s configuration, this could be a login at the console (that is, the keyboard and monitor that are directly connected to the computer) or remotely (via serial line, modem, or network). When an individual logs in, that person may use some or all of the programs and resources available on the computer. Some other resources, like files delivered by a Web server, don’t require a login.

Password protection Linux accounts are protected by passwords. A person attempting to log in must provide both a username and a password. The username is generally public knowledge, but the password is secret. Some forms of login bypass the password protection, usually by deferring to authentication performed by another computer.

Permissions Every account has permission to run certain programs and access certain files. These permissions are controlled on a file-by-file basis, as described in Chapter 2, “Using Text-Mode Commands.”

Home directory Every account has a home directory associated with it. This is a directory in which the user can store data files. Typically, each user has his or her own home directory, although it’s possible to configure a system so that two or more users share a home directory. It’s also possible, but seldom useful, to specify a home directory to which a user cannot write. (You might use such a configuration if a user should be able to run programs that don’t generate their own data but should not be able to store files on the computer.)

User and group IDs Computers operate on numbers, not words—the words we see on computer screens are encoded as numbers internally. Linux associates two numbers with each account. The first is the *user ID (UID)*, which is mapped to a specific username. The second is the *group ID (GID)*, which is mapped to a specific group of users. Both these processes are described further in the section “Mapping UIDs and GIDs to Users and Groups.”

Default shell When using a Linux computer at a text-based login (say, at the console without the X Window System running, or via a text-based network protocol like Telnet), Linux presents users with a program known as a *shell*, as described in Chapter 2. Several shells are available for Linux and can be set on an account-by-account basis.

Program-specific files Some programs generate files that are associated with a particular user, in or out of that user's home directory. Many programs create configuration files in the user's home directory, for instance. Another important example is the mail spool, in which a Linux system stores incoming e-mail messages for a user. Assuming the basic mail software is installed, creating a user account is usually necessary and sufficient for a user to receive mail, although exceptions to this rule exist, particularly with some mail server packages.

Some of these features are defined in one or two critical system configuration files: `/etc/passwd` and `/etc/shadow`. The `/etc/passwd` file is the traditional repository for critical account information, including the username, UID number, GID number, password, home directory location, and default shell specification. Creating or modifying an account is mostly a matter of modifying this one file. There are enough additional details, though, that most administrators use special tools to perform these tasks, as described in the section "Configuring User Accounts."

Unfortunately, the needs of the system dictate that `/etc/passwd` be readable by all users. This fact makes the placement of password information in `/etc/passwd`—even in encrypted form—a risky proposition. For this reason, most Linux distributions since the late 1990s ship with *shadow password* support. In this system, users' passwords are stored in a separate file, `/etc/shadow`. This file cannot be read by most users, making it more difficult for a miscreant with an account on the computer to break into other users' accounts.

Accounts in a Multitasking System

Linux is both a multiuser and a multitasking system. Linux's multiuser nature allows multiple people to use one computer without causing problems for one another. Linux's multitasking ability allows multiple programs to run at one time. Although single-user multitasking OSs are available, combining the two has many advantages, particularly in a networked environment. Specifically, several people can be logged onto a Linux computer at one time, and they can run the same or different programs simultaneously. For instance, Sally can run the Emacs editor while Sam and Ellen both run the Mozilla Web browser and George runs a C compiler.

Although it's possible to use a single account for multiple simultaneous logins, using multiple accounts can be helpful, particularly when multiple individuals are involved. Each account can be configured with its owner's preferences in mind, and therefore, simultaneous logins can present different defaults for things like the placement of icons on a desktop environment or the command shell to be used. Furthermore, if a user changes a default value, that change will not affect other users currently logged on to the computer. If the system were a single-user computer that allowed multiple logins, changes to system defaults could adversely affect other users or be undone when other users logged out.

Of course, Linux's multitasking ability doesn't mean that the computer can support an unlimited number of simultaneous users. Some activities, such as George's C program compilation, are likely to consume a great deal of RAM, CPU time, or disk I/O. If many users try to run such resource-intensive programs simultaneously, all the users will see a performance decrease. Just how many simultaneous users a Linux computer can support depends on many factors, including the types of programs they're likely to run and how much of critical system resources (RAM, CPU speed, network speed, disk speed, and disk capacity)

the system has. If the applications used aren't very resource intensive, a single modern computer can support dozens or hundreds of simultaneous users, but if the programs are hogs of one or more resources, one user per computer may seem like too many.

Simultaneous use of one computer by multiple users generally requires some form of network connectivity, although it can also be handled through terminals connected to serial ports. Typically, remote login protocols like Telnet or the Secure Shell (SSH) support text-mode logins. Linux's GUI environment, the X Window System (or X for short), is network-enabled, so it permits remote use of GUI programs. Alternatively, the VNC program (<http://www.realvnc.com>) supports similar connectivity.

Linux supports multiple simultaneous logins through its standard console via a feature known as *virtual terminals* (VTs). From a text-mode login, hitting the Alt key along with a function key from F1 to F6 typically switches to a different virtual screen, and you can log into as many of these as you like. You can even run multiple X sessions at different resolutions by issuing appropriate parameters to `startx`. Ordinarily, the first X session runs on VT 7, although some distributions use other VTs for this purpose, such as VT 1 or VT 9. When switching out of a VT that's running X, you must add Ctrl to the key sequence—for instance, you must press Ctrl+Alt+F1 to switch from X to the first text-mode VT. You can run a second X session by logging into a text VT and issuing the following command:

```
$ startx -- :1 vt8
```

This command will run X in VT 8. You can switch back and forth between it and the first X session by pressing Ctrl+Alt+F7 and Ctrl+Alt+F8.

Of course, this VT capability is most useful for a single-user workstation—two people can't make practical use of the same keyboard at the same time. Nonetheless, it's still useful if you as an administrator want to run Linux under multiple accounts or X configurations, or if you want to easily switch between multiple text-based programs without running X.

The Superuser Account

One particularly important account on all Linux systems is that of the *superuser*. The superuser is also referred to as the administrator. The account used by the superuser is known as `root`.

Whenever you perform system administration tasks on a Linux computer, you'll do so as `root`. You can do this in any of several ways:

Direct administrative login You can log into the computer as `root`. Thereafter, any action you perform will be done as the superuser. This can be a very dangerous way to use the system, so it's best to do so only for brief periods. Most systems contain restrictions on `root` logins, so they can be done only from the console. This helps prevent outsiders from gaining access to a system over a network by using a stolen password.

Switching identities after login The `su` program lets you temporarily acquire superuser privileges or take on any other user's identity. Type `su` and press the Enter key after logging on as an ordinary user, and the system will prompt you for the `root` password. If you type that password correctly, subsequent commands will be executed as `root`. Type `exit` to return to your

normal user privileges. To take on a non-root user's privileges, add that user's name, as in **su george**, to take on the george account's role. If you're already **root**, you can take on another user's identity without that user's password; **su** doesn't ask **root** for a password. This can be useful when you're debugging problems that may be related to a particular user's configuration.

Running an individual program as the superuser Once configured, the **sudo** command allows you to execute a single command as **root**. This limits the danger of running as **root**, so it can be a good way to run the programs that you most frequently run as **root**. The **/etc/sudoers** file contains a list of users who may use **sudo**, as well as the commands they are allowed to run in this way. You can edit this file with the **visudo** command, which invokes the Vi editor (as described in Chapter 3, “Managing Processes and Editing Files”) in such a way that it helps you get the format of the configuration file right. To use **sudo**, you type this command followed by the command you want to execute, as in **sudo fdisk /dev/hda** to edit the partition table on **/dev/hda** without using **su** or some other method of acquiring **root** privileges. The first time you run **sudo** in a given shell, or if you use the program after you've not used it for a while, you'll be asked for a password. This is normally your ordinary user password, but some configurations require you to enter the **root** password.

SUID root files As described in the section “Interpreting File Access Codes” in Chapter 2, it's possible to set a file to execute as if run by **root** even when it's run by another user. This feature must be set on a program-by-program basis.

Program prompts Some configuration tools prompt you for the **root** password and then run themselves as **root**. This setup is most common with the GUI configuration tools that ship with many Linux distributions.

The Danger of **root** Power

The **root** account is special because it bypasses normal security features. Specifically, the superuser may read, write, or delete any file on the computer, no matter who owns that file or whether the owner has granted other users read or write access to it. This sort of power is dangerous not just because of the ability to invade other users' privacy, but because it allows **root** to do serious damage to the OS. For instance, suppose you want to delete a directory and its contents. You might issue the following command to do so:

```
# rm -r /home/george/olddir
```

This command deletes the **/home/george/olddir** directory and all its files and subdirectories. Unfortunately, a single typo can create a much more destructive command:

```
# rm -r / home/george/olddir
```

Note the stray space between **/** and **home/george/olddir**. This typo causes the computer to delete all files in the **/** directory—that is, all files on the computer, not just the files in **home/george/olddir**. This is the sort of power that you should grant yourself only when you absolutely need it.

Linux Usernames

Linux is fairly flexible about its usernames. Most versions of Linux support usernames consisting of any combination of upper- and lowercase letters, numbers, and many punctuation symbols, including periods and spaces. Some punctuation symbols, however, such as spaces, cause problems for certain Linux utilities, so it's generally best to avoid using punctuation in Linux usernames. Underscores (`_`) and periods (`.`) are relatively unlikely to cause problems and so are occasionally used. Also, usernames must begin with a letter, so a username such as `45u` is invalid, although `u45` is fine. Although usernames may consist of up to 32 characters, many utilities truncate usernames longer than 8 characters or so in their displays, so many administrators try to limit username length to 8 characters.

Linux treats usernames in a case-sensitive way. Therefore, a single computer can support both `ellen` and `Ellen` as separate users. This practice can lead to a great deal of confusion, however, so it's best to avoid creating accounts whose usernames differ only in case. In fact, the traditional practice is to use entirely lowercase letters in Linux usernames, such as `sally`, `sam`, `ellen`, and `george`. Usernames don't need to be based on first names, of course—you could use `sam_jones`, `s.jones`, `sjones`, `jones`, `jones17`, or `u238`, to name just a few possibilities. Most sites develop a standard method of creating usernames, such as using the first initial and the last name. Creating and following such a standard practice can help you locate an account that belongs to a particular individual. If your computer has many users, though, you may find a naming convention produces duplicates, particularly if your standard uses initials to shorten usernames. You may therefore be forced to deviate from the standard or incorporate numbers to distinguish between all the Davids or Smiths of the world.

Linking Users Together for Productivity via Groups

Linux uses *groups* as a means of organizing users. In many ways, groups parallel users. Groups are similar to users in several ways:

- Groups are defined in a single file, `/etc/group`, which has a structure similar to that of `/etc/passwd`.
- Groups have names similar to usernames.
- Group names are tied to group IDs (GIDs).

Groups are *not* accounts, however. Rather, groups are a means of organizing collections of accounts, largely as a security measure. As described in Chapter 2, every file on a Linux system is associated with a specific group, and various permissions can be assigned to members of that group. For instance, group members (such as faculty at a university) might be allowed to read a file, but others (such as students) might be disallowed such access. Because Linux provides access to most hardware devices (such as serial ports and tape backup units) through files, this same mechanism can be used to control access to hardware.

Every group has anywhere from no members to as many members as there are users on the computer. Group membership is controlled through the `/etc/group` file. This file contains a list of groups and the members belonging to each group. The details of this file's contents are described in the section “Configuring Groups.”

In addition to membership defined in `/etc/group`, each user has a default or primary group. The user's primary group is set in the user's configuration in `/etc/passwd`. When users log onto the computer, their group membership is set to their primary groups. When users create files or launch programs, those files and running programs are associated with a single group—the current group membership. A user can still access files belonging to other groups, as long as the user belongs to that group and the group access permissions allow the access. To run programs or create files with other than the primary group membership, however, the user must run the `newgrp` command to switch current group membership. For instance, to change to the `project2` group, you might type the following:

```
$ newgrp project2
```

If the user typing this command is listed as a member of the `project2` group in `/etc/group`, the user's current group membership will change. Thereafter, files created by that user will be associated with the `project2` group. Alternatively, users can change the group associated with an existing file by using the `chgrp` or `chown` command, as described in Chapter 2.

This group structure enables you to design a security system that permits different collections of users to easily work on the same files while simultaneously keeping other users of the same computer from prying into files they should not be able to access. In a simple case, you might create groups for different projects, classes, or workgroups, with each user restricted to one of these groups. A user who needs access to multiple groups could be a member of each of these groups—for instance, a student who takes two classes could belong to the groups associated with each class, or a supervisor might belong to all the supervised groups. The section “Using Common User and Group Strategies” describes the approaches taken by various Linux distributions by default, and it then explains how you can expand and use these strategies to suit your own needs.

Mapping UIDs and GIDs to Users and Groups

As mentioned earlier, Linux defines users and groups by numbers (UIDs and GIDs, respectively). Internally, Linux tracks users and groups by these numbers, not by name. For instance, the user `sam` might be tied to UID 523, and `ellen` might be UID 609. Similarly, the group `project1` might be GID 512, and `project2` might be GID 523. For the most part, these details take care of themselves—you use names, and Linux uses `/etc/passwd` or `/etc/group` to locate the number associated with the name. You may occasionally need to know how Linux assigns numbers when you tell it to do something, though. This is particularly true when you are troubleshooting or if you have cause to manually edit `/etc/passwd` or `/etc/group`.

Linux distributions reserve the first 100 user and group IDs (0–99) for system use. The most important of these is 0, which corresponds to `root` (both the user and the group). Subsequent low numbers are used by accounts and groups that are associated with specific Linux utilities and functions. For instance, UID 2 and GID 2 are generally the `daemon` account and group, respectively, which are used by various servers; and UID 8 and GID 12 are usually

the mail account and group, which can be used by mail-related servers and utilities. Not all account and group numbers from 0 to 99 are in use; there are usually only one or two dozen accounts and a dozen or so groups used in this way. You can check your `/etc/passwd` and `/etc/group` files to determine which user and group IDs are so used.



Aside from UID 0 and GID 0, UID and GID numbers aren't fully standardized. For instance, although UID 2 and GID 2 map to the daemon account and daemon group on Fedora, on Ubuntu UID 2 and GID 2 map to the bin account and bin group; the daemon account and group correspond to UID 1 and GID 1. If you need to refer to a particular user or group, use the name rather than the number.

Beyond 100, user and group IDs are available for use by ordinary users and groups. Many distributions, however, reserve up to 500 or even 1000 for special purposes. Frequently, therefore, the first normal user account is assigned a UID of 500 or 1000. When you create additional accounts, the system typically locates the next-highest unused number, so the second user you create is UID 501, the third is 502, and so on. When you remove an account, that account's ID number may be reused, but the automatic account-creation tools typically don't do so if subsequent numbers are in use, leaving a gap in the sequence. This gap causes no harm unless you have so many users that you run out of ID numbers. (The limit is 65,536 users with the 2.2.x kernels and more than 4.2 billion with the 2.4.x and later kernels, including root and other system accounts. The limit can be set lower in configuration files or because of limits in support programs.) In fact, reusing an ID number can cause problems if you don't clear away the old user's files—the new user will become the owner of the old user's files, which can lead to confusion.

Typically, GID 100 is users—the default group for some distributions. (See “Using Common User and Group Strategies” later in this chapter.) On any but a very small system with few users, you'll probably want to create your own groups. Because different distributions have different default ways of assigning users to groups, it's best that you familiarize yourself with your distribution's way of doing this and plan your own group-creation policies with this in mind. For instance, you might want to create your own groups within certain ranges of IDs to avoid conflicts with the distribution's default user- and group-creation processes.

It's possible to create multiple usernames that use the same UID, or multiple group names that use the same GID. In some sense, these are different accounts or groups; they have different entries in `/etc/passwd` or `/etc/group`, so they can have different home directories, different passwords, and so on. Because these users or groups share IDs with other users or groups, though, they're treated identically in terms of file permissions. Unless you have a compelling reason to do so, you should avoid creating multiple users or groups that share an ID.



Intruders sometimes create accounts with UID 0 to give themselves root privileges on the systems they invade. Any account with a UID of 0 is effectively the root account, with all the power of the superuser. If you spot a suspicious account in your `/etc/passwd` file with a UID of 0, your system has probably been compromised.



Real World Scenario

Coordinating UIDs and GIDs Across Systems

If you maintain several Linux computers and want to set up Network Filesystem (NFS) file sharing, one problem that can arise is keeping UIDs and GIDs synchronized across systems. Because all Linux filesystems, including NFS, track numeric IDs rather than the names that humans use, mismatched UIDs and GIDs can cause one person's files to appear to be owned by another person on an NFS mount. For instance, suppose that two computers each have two users, ellen and george. On one computer, ellen has UID 500 and george has UID 501, but these numbers are reversed on the other. As a consequence, when one computer mounts the other's files via NFS, the UID values will indicate that ellen owns files that are really owned by george, and vice versa.

One solution to this problem is to keep UIDs and GIDs consistent across computers. This isn't too difficult with a handful of small systems with few users, but it becomes tedious with larger or more systems. Some versions of the Linux NFS clients and servers also support various mapping options, such as using a static map file or using a user ID mapping server run on the client system. Unfortunately, these options are no longer being actively supported. Another option is to use a centralized login database, such as one maintained via the Network Information System (NIS) or the Lightweight Directory Access Protocol (LDAP), to coordinate accounts on multiple computers. Chapter 12, "Securing Linux," describes the basics of NIS and LDAP configuration.

Understanding Home Directories

A user's *home directory* is a directory on the disk that's usually intended for one user alone. On Linux systems, the standard placement of home directories is in the /home directory tree, with each user's home directory named after the user's account name. For instance, the home directory for the sally account would be /home/sally. This naming and placement is only a convention, though—it's not a requirement. The /etc/passwd file contains the location of each user's home directory, so you can modify this location by editing that file. You can also specify an alternative location when you create an account (as described shortly in the section "Adding Users"), or use the usermod utility to change it after the fact.

Typically, a user's home directory belongs to that user only. Therefore, it's created with fairly restrictive permissions, particularly for writing to the directory. The exact permissions used by default vary from one distribution to another, so you should check yours to see how it's done. If you want to create more stringent (or more lax) permissions, you'll have to do so yourself after creating an account, or you'll need to create your own account-creation scripts to automate the process.

You can create separate directories for shared projects, if you like. For instance, you might want to have a directory in which group members can store files that belong to the

group as a whole, or in which group members may exchange files. Linux distributions don't create such directories automatically when creating groups, so you'll have to attend to this task yourself, as well as decide where to store them. (Somewhere in `/home` is a logical choice, but it is up to you.)

One problem that's commonly faced by Linux system administrators is the depletion of available disk space. The `/home` directory frequently resides on a separate partition, and sometimes an entirely separate physical hard disk, from other Linux files. This arrangement can make the system more secure because it helps to isolate the data—filesystem corruption on one partition need not affect data on another. It also limits room for expansion, however. If your users begin creating very large files or if the number of users you must support grows and causes your initial estimates of required `/home` disk space to be exceeded, you'll need to take action to correct this matter. For instance, you might move home directories to some other partition, enlarge the home partition with a tool like GNU Parted (<http://www.gnu.org/software/parted/>), or add a new hard disk to store some or all of the user home directories.

Configuring User Accounts

How frequently you'll do user maintenance depends on the nature of the system you administer. Some systems, such as small personal workstations, will need changes very rarely. Others, such as large systems in environments in which users are constantly coming and going, may require daily maintenance. The latter situation would seem to require more knowledge of user account configuration tools, but even if you administer a seldom-changing system, it's useful to know how to do these things so that you can do them quickly and correctly when you do need to add, modify, or delete user accounts.

Adding Users

You can add users through the `useradd` utility. (This program is called `adduser` on some distributions.) Its basic syntax is as follows:

```
useradd [-c comment] [-d home-dir] [-e expire-date] [-f inactive-days]
[-g initial-group] [-G group[,...]] [-m [-k skeleton-dir] | -M]
[-p password] [-s shell] [-u UID [-o]] [-r] [-n] username
```



Some of these parameters modify settings that are valid only when the system uses shadow passwords. This is the standard configuration for most distributions today.

In its simplest form, you may type just **`useradd username`**, where *username* is the username you want to create. The rest of the parameters are used to modify the default values for the system, which are stored in the file `/etc/login.defs`.

The parameters for the `useradd` command modify the program’s operation in various ways, as summarized in Table 5.1.

TABLE 5.1 Options for `useradd`

Option Name	Option Abbreviation	Effect
<code>--comment</code> <i>comment</i>	<code>-c</code>	This parameter specifies the comment field for the user. Some administrators store public information such as a user’s office or telephone number in this field. Others store just the user’s real name or no information at all.
<code>--home</code> <i>home-dir</i>	<code>-d</code> (or <code>-h</code> on some versions of <code>useradd</code>)	You specify the account’s home directory with this parameter. It defaults to <code>/home/username</code> on most systems.
<code>--expiredate</code> <i>expire-date</i>	<code>-e</code>	You set the date on which the account will be disabled, expressed in the form <code>YYYY-MM-DD</code> , with this option. (Many systems will accept alternative forms, such as <code>MM-DD-YYYY</code> , or a single value representing the number of days since January 1, 1970.) The default is for an account that does not expire. This option is most useful in environments in which user accounts are inherently time-limited, such as accounts for students taking particular classes or temporary employees.
<code>--inactive</code> <i>inactive-days</i>	<code>-f</code>	This parameter sets the number of days after a password expires after which the account becomes completely disabled. A value of <code>-1</code> disables this feature and is the default.
<code>--gid</code> <i>default-group</i>	<code>-g</code>	You set the name or GID of the user’s default group with this option. The default for this value varies from one distribution to another, as described later, in “Using Common User and Group Strategies.”
<code>--groups</code> <i>group[,...]</i>	<code>-G</code>	This parameter sets the names or GIDs of one or more groups to which the user belongs. These groups need not be the default group, and more than one may be specified by separating them with commas.
<code>--create-home</code>	<code>-m</code>	When this option is included, <code>useradd</code> creates a home directory for the user. Many distributions use <code>-m</code> as the default when running <code>useradd</code> .

TABLE 5.1 Options for useradd (continued)

Option Name	Option Abbreviation	Effect
--skeleton-dir skeleton-dir	-k	Normally, default configuration files are copied from /etc/skel, but you may specify another template directory with this option, which is valid only in conjunction with -m.
None	-M	This option forces the system to <i>not</i> automatically create a home directory, even if /etc/login.defs specifies that this action is the default.
--password encrypted-password	-p	This parameter passes the <i>preencrypted</i> password for the user to the system. The <i>encrypted-password</i> value will be added, <i>unchanged</i> , to the /etc/passwd or /etc/shadow file. This means that if you type an unencrypted password, it won't work as you probably expected. In practice, this parameter is most useful in scripts, which can encrypt a password (using crypt) and then send the encrypted result through useradd. The default value disables the account, so you must run passwd to change the user's password.
--shell shell	-s	You set the name of the user's default login shell with this option. On most systems, this defaults to /bin/bash, but Linux supports many alternatives, such as /bin/tcsh and /bin/zsh.
--uid UID	-u	This parameter creates an account with the specified user ID value (<i>UID</i>). This value must be a positive integer, and it is normally 500 or above for user accounts. System accounts typically have numbers below 100.
--non-unique	-o	This parameter enables a single UID number to be reused; this option is passed when creating the second or subsequent account that reuses a UID.
--system	-r	This parameter specifies the creation of a system account—an account with a value lower than UID_MIN, as defined in /etc/login.defs. (This is normally 100, 500, or 1000.) useradd also doesn't create a home directory for system accounts.
--no-user-group	-n	In some distributions, such as Red Hat, the system creates a group with the same name as the specified username. This parameter disables this behavior.



Some versions of `useradd`, and other utilities described in this chapter, support only the one-character options in the “Option Abbreviation” columns in this chapter.

Suppose you’ve added a new hard disk in which some users’ home directories are located and mounted it as `/home2`. You want to create an account for a user named Sally in this directory and make the new user a member of the `project1` and `project4` groups, with default membership in `project4`. The user has also requested `tcsh` as her default shell. You might use the following commands to accomplish this goal:

```
# useradd -d /home2/sally -g project4 -G project1,project4 -s /bin/tcsh sally
# passwd sally
Changing password for user sally
New UNIX password:
Retype new UNIX password:
passwd: all authentication tokens updated successfully
```



The `passwd` command asks for the password twice, but it does not echo what you type. This prevents somebody who sees your screen from reading the password off it. The details of the program’s prompts and messages vary from one system to another. `passwd` is described in more detail in the next section.

Modifying User Accounts

You can modify user accounts in many ways. You can directly edit critical files such as `/etc/passwd`, modify user-specific configuration files in the account’s home directory, or use system utilities like those used to create accounts. You usually modify an existing user’s account at the user’s request or to implement some new policy or system change, such as moving home directories to a new hard disk. Sometimes, though, you must modify an account immediately after its creation, in order to customize it in ways that aren’t easily handled through the account-creation tools or because you realize you forgot a parameter to `useradd`.

Setting a Password

Although `useradd` provides the `-p` parameter to set a password, this tool is not very useful when directly adding a user because it requires a preencrypted password. Therefore, it’s usually easiest to create an account in disabled form (by not using `-p` with `useradd`) and set the password after creating the account. This can be done with the `passwd` command, which has the following syntax:

```
passwd [-k] [-l] [-u [-f]] [-d] [-S] [username]
```


The parameters to this command enable you to modify its behavior, as summarized in Table 5.2.

TABLE 5.2 Options for passwd

Option Name	Effect
-k	This parameter indicates that the system should update an expired account.
-l	This parameter locks an account by prefixing the encrypted password with an exclamation mark (!). The result is that the user can no longer log into the account, but the files are still available and the change can be easily undone.
-u	This parameter unlocks an account by removing a leading exclamation mark. <code>useradd</code> creates accounts that are locked and have no password, so using this command on a fresh account would result in an account with no password. Normally, <code>passwd</code> doesn't allow this—it returns an error if you attempt it.
-f	Adding this parameter to <code>-u</code> forces <code>passwd</code> to turn an account into one with no password.
-d	This parameter removes the password from an account, rendering it password-less.
-S	You can have <code>passwd</code> display information on the password for an account—whether or not it's set and what type of encryption it uses—with this option.

Ordinary users can use `passwd` to change their passwords, but many `passwd` parameters can be used only by `root`. Specifically, `-l`, `-u`, `-f`, `-d`, and `-S` are all off-limits to ordinary users. Similarly, only `root` can specify a username to `passwd`. When ordinary users run the program, they should omit their usernames, and `passwd` will change the password for the user who ran the program. As a security measure, `passwd` asks for a user's old password before changing the password when an ordinary user runs the program. This precaution is *not* taken when `root` runs the program so that the superuser can change a user's password without knowing the original password. Since the administrator normally doesn't know the user's password, this is necessary.

Linux passwords may consist of letters, numbers, and punctuation. Linux distinguishes between upper- and lowercase letters in passwords, which means you can use mixed-case passwords to improve security.



The section “Improving Account Security” later in this chapter includes additional information on selecting good passwords.

Using *usermod*

The *usermod* program closely parallels *useradd* in its features and parameters, as summarized in Table 5.2. This utility changes an existing account instead of creating a new one, however. The major differences between *useradd* and *usermod* are as follows:

- *usermod* allows the addition of a *-m* parameter when used with *-d*. The *-d* parameter alone changes the user’s home directory, but it does not move any files. Adding *-m* causes *usermod* to move the user’s files to the new location.
- *usermod* supports a *-l* parameter, which changes the user’s login name to the specified value. For instance, typing ***usermod sally -l sjones*** changes the username from *sally* to *sjones*.
- You may lock or unlock a user’s password with the *-L* and *-U* options, respectively. This duplicates functionality provided by *passwd*.

The *usermod* program changes the contents of */etc/passwd* or */etc/shadow*, depending on the option used. If *-m* is used, *usermod* also moves the user’s files, as already noted.



Changing an account’s characteristics while the owner is logged in can have undesirable consequences. This is particularly true of the *-d -m* combination, which can cause the files a user was working on to move. Most other changes, such as changes to the account’s default shell, simply don’t take effect until the user has logged out and back in again.

If you change the account’s UID, this action does *not* change the UIDs stored with a user’s files. Because of this, the user may lose access to these files. You can manually update the UIDs on all files by using the *chown* command (see the section “Changing File Ownership and Permissions” in Chapter 2). Specifically, a command like the following, issued after changing the UID on the account *sally*, will restore proper ownership on the files in *sally*’s home directory:

```
# chown -R sally /home/sally
```

This action does *not* change the ownership of files that aren’t in *sally*’s home directory. If you believe such files exist, you may need to track them down with the *find* command, as you’ll see in the upcoming section “Deleting Accounts.” Also, this command blindly changes ownership of *all* files in the */home/sally* directory. This is probably desirable, but it’s conceivable that some files in that directory *should* be owned by somebody else—say, because *sally* and another user are collaborating on a project.

When using the *-G* option to add a user to new groups, be aware that any groups *not* listed will be removed. The *gpasswd* command, described in the upcoming section, “Using *gpasswd*,” provides a way to add a user to one or more specific groups without affecting existing group memberships, and therefore it is generally preferable for this purpose.

Using *chage*

The *chage* command enables you to modify account settings relating to account expiration. It's possible to configure Linux accounts so that they automatically expire if either of two conditions is true:

- The password hasn't been changed in a specified period of time.
- The system date is past a predetermined time.

The first option is generally used to enforce password changes—say, to get users to change their passwords once a month. The second option is useful when an account should exist for a specific limited period of time, such as until the end of an academic semester or until a temporary employee leaves. These settings are controlled through the *chage* utility, which has the following syntax:

```
chage [-l] [-m mindays] [-M maxdays] [-d lastday] [-I inactivedays]
[-E expiredate] [-W warndays] username
```

The program's parameters modify the command's actions, as summarized in Table 5.3.

TABLE 5.3 Options for *chage*

Option Name	Option Abbreviation	Effect
--list	-l	This option causes <i>chage</i> to display account expiration and password aging information for a particular user.
--mindays <i>days</i>	-m	This parameter sets the minimum number of days between password changes. 0 indicates that a user can change a password multiple times in a day, 1 means that a user can change a password once a day, 2 means that a user may change a password once every two days, and so on.
--maxdays <i>days</i>	-M	This parameter sets the maximum number of days that may pass between password changes. For instance, 30 would require a password change approximately once a month. If the user changes a password before this deadline, the counter is reset to the password change date.
--lastday <i>date</i>	-d	This parameter sets the last day a password was changed. This value is normally maintained automatically by Linux, but you can use this parameter to artificially alter the password change count. For instance, you could use this to set the last changed date to force a password change in some period of time you determine. <i>date</i> is expressed in the format <i>YYYY-MM-DD</i> , or as the number of days since January 1, 1970.

TABLE 5.3 Options for `chage` (continued)

Option Name	Option Abbreviation	Effect
<code>--inactive days</code>	<code>-I</code>	This parameter sets the number of days between password expiration and account disablement. An expired account may not be used or may force the user to change the password immediately upon logging in, depending on the distribution. A disabled account is completely disabled, however.
<code>--expiredate date</code>	<code>-E</code>	You can set an absolute expiration date with this option. For instance, you might use <code>-E 2010-05-21</code> to have an account expire on May 21, 2010. The date may also be expressed as the number of days since January 1, 1970. A value of <code>-1</code> represents no expiration date.
<code>--warndays days</code>	<code>-W</code>	This option sets the number of days before account expiration that the system will warn the user of the impending expiration. It's generally a good idea to use this feature to alert users of their situation, particularly if you make heavy use of password change expirations.

The `chage` command can normally be used only by root. The one exception to this rule is if the `-l` option is used; this feature allows ordinary users to check their account expiration information.

Directly Modifying Account Configuration Files

You can modify user configuration files directly. `/etc/passwd` and `/etc/shadow` control most aspects of an account's basic features, but many files within a user's home directory control user-specific configuration; for instance, `.bashrc` can be used to set user-specific Bash features. This latter class of configuration files is far too broad to cover here, but `/etc/passwd` and `/etc/shadow` are not. Both files consist of a set of lines, one line per account. Each line begins with a username and continues with a set of fields, delimited by colons (:). Many of these items may be modified with `usermod` or `passwd`. A typical `/etc/passwd` entry resembles the following:

```
sally:x:529:100:Sally Jones:/home/sally:/bin/bash
```

Each field has a specific meaning:

Username The first field in each `/etc/passwd` line is the username (sally in this example).

Password The second field has traditionally been reserved for the password. Most Linux systems, however, use a shadow password system in which the password is stored in

`/etc/shadow`. The `x` in the example's password field is an indication that shadow passwords are in use. In a system that does not use shadow passwords, an encrypted password will appear here instead.

UID Following the password is the account's user ID (529 in this example).

Primary GID The default login group ID is next in the `/etc/passwd` line for an account. The example uses a primary GID of 100.

Comment The comment field may have different contents on different systems. In the preceding example, it's the user's full name. Some systems place additional information here, in a comma-separated list. Such information might include the user's telephone number, office number, title, and so on.

Home directory The user's home directory is next up in the list.

Default shell The default shell is the final item on each line in `/etc/passwd`. This is normally `/bin/bash`, `/bin/tcsh`, or some other common command shell. It's possible to use something unusual here, though. For instance, many systems include a shutdown account with `/bin/shutdown` as the shell. If you log into this account, the computer immediately shuts down. You can create user accounts with a shell of `/bin/false`, which prevents users from logging in as ordinary users but leaves other utilities intact. Users can still receive mail and retrieve it via a remote mail retrieval protocol like POP or IMAP, for instance. A variant on this scheme uses `/bin/passwd` so that users may change their passwords remotely but may not log in using a command shell.

You can directly modify any of these fields, although in a shadow password system, you probably do *not* want to modify the password field; you should make password-related changes via `passwd` so that they can be properly encrypted and stored in `/etc/shadow`. As with changes initiated via `usermod`, it's best to change `/etc/passwd` directly only when the user in question isn't logged in, to prevent a change from disrupting an ongoing session.

Like `/etc/passwd`, `/etc/shadow` may be edited directly. An example `/etc/shadow` line resembles the following:

```
sally:E/moFkeT5UnTQ:14343:0:-1:7:-1:-1:
```

Most of these fields correspond to options set with the `chage` utility, although some are set with `passwd`, `useradd`, or `usermod`. The meaning of each colon-delimited field of this line is as follows:

Username Each line begins with the username. Note that the UID is *not* used in `/etc/shadow`; the username links entries in this file to those in `/etc/passwd`.

Password The password is stored in encrypted form, so it bears no obvious resemblance to the actual password. An asterisk (*) or exclamation mark (!) denotes an account with no password (that is, the account doesn't accept logins—it's locked). This is common for accounts used by the system itself.



If you've forgotten the root password for a system, you can boot with an emergency recovery system and copy the contents of a password field for an account whose password you do remember. You can then boot normally, log in as root, and change the password. In a real pinch, you can delete the contents of the password field, which results in a root account with *no* password (that is, none is required to log in). Be *sure* to *immediately* change the root password after rebooting if you do this, though!

Last password change The next field (14343 in this example) is the date of the last password change. This date is stored as the number of days since January 1, 1970.

Days until change allowed The next field (0 in this example) is the number of days before a password change is allowed.

Days before change required This field is the number of days after the last password change before another password change is required.

Days warning before password expiration If your system is configured to expire passwords, you may set it to warn the user when an expiration date is approaching. A value of 7, as in the preceding example, is typical.

Days between expiration and deactivation Linux allows for a gap between the expiration of an account and its complete deactivation. An expired account either cannot be used or requires that the user change the password immediately after logging in. In either case, its password remains intact. A deactivated account's password is erased, and the account cannot be used until it's reactivated by the system administrator.

Expiration date This field shows the date on which the account will be expired. As with the last password change date, the date is expressed as the number of days since January 1, 1970.

Special flag This field is reserved for future use and is normally not used or contains a meaningless value. This field is empty in the preceding example.

For fields relating to day counts, values of -1 or 99999 typically indicate that the relevant feature has been disabled. The `/etc/shadow` values are generally best left to modification through the `usermod` or `chage` commands because they can be tricky to set manually—for instance, it's easy to forget a leap year or the like when computing a date as the number of days since January 1, 1970. Similarly, because of its encrypted nature, the password field cannot be edited effectively except through `passwd` or similar utilities. (You could cut and paste a value from a compatible file or use `crypt` yourself, but it's usually easier to use `passwd`. Copying encrypted passwords from other systems is also somewhat risky because it means that the users will have the same passwords on both systems, and this fact will be obvious to anybody who's acquired both encrypted password lists.)



/etc/shadow is normally stored with very restrictive permissions, such as `rw-----`, with ownership by root. This fact is critical to the shadow password system's utility since it keeps non-root users from reading the file and obtaining the password list, even in an encrypted form. Therefore, if you manually modify /etc/shadow, be sure it has the correct permissions when you're done.



Real World Scenario

Network Account Databases

Many networks employ network account databases. Such systems include the Network Information System (NIS), an update to this system called NIS+, the Lightweight Directory Access Protocol (LDAP), Kerberos realms, Windows NT 4.0 domains, and Active Directory (AD) domains. All of these systems move account database management onto a single centralized computer (often with one or more backup systems). The advantage of this approach to account maintenance is that users and administrators need not deal with maintaining accounts independently on multiple computers. A single account database can handle accounts on dozens (or even hundreds or thousands) of different computers, greatly simplifying day-to-day administrative tasks and simplifying users' lives. Using such a system, though, means that most user accounts won't appear in /etc/passwd and /etc/shadow, and groups may not appear in /etc/group. (These files will still hold information on local system accounts and groups, though.)

Linux can participate in these systems, naturally. Activating use of such network account databases after installing Linux is a complex topic. Chapter 12 covers a few basics for NIS and LDAP. My book *Linux in a Windows World* (O'Reilly, 2005) covers this topic for Windows NT 4.0 domains, LDAP, and Kerberos.

Deleting Accounts

On the surface, deleting user accounts is easy. You may use the `userdel` command to do the job of removing a user's entries from /etc/passwd and, if the system uses shadow passwords, /etc/shadow. The `userdel` command takes just three parameters: `--remove` (or `-r`) causes the system to remove all files from the user's home directory, as well as the home directory itself; `--help` (or `-h`) displays help information; and `--force` (`-f`) forces account removal even if the user is currently logged in. Thus, removing a user account such as `sally` is easily accomplished with the following command:

```
# userdel -r sally
```

You can omit the `-r` parameter if you want to preserve the user's files. There is one potential complication, however: users might create files *outside* their home directories. For instance, many programs use the `/tmp` directory as “scratch space,” so user files often wind up there. These files will be deleted automatically after a certain period, but you may have other directories in which users might store files. To locate all such files, you can use the `find` command with its `-uid` parameter. For instance, if `sally` had been UID 529, you might use the following command to locate all her files:

```
# find / -uid 529
```

The result will be a list of files owned by UID 529 (formerly `sally`). You can then go through this list and decide what to do with the files—change their ownership to somebody else, delete them, back them up to a removable disk, or what have you. It's wise to do *something* with these files, though, or else they may be assigned ownership to another user if Sally's UID is reused. This could become awkward if the files exceed the new user's disk quota or if they contain information that the new user should not have—such a person might mistakenly be accused of indiscretions or even crimes.

A few servers—most notably Samba—keep their own lists of users. If you run such a server, it's best to remove the user's listing from that server's user list when you remove the user's main account. In the case of Samba, this is normally done by manually editing the `smbpasswd` file (usually located in `/etc`, `/etc/samba`, or `/etc/samba.d`) and deleting the line corresponding to the user in question or by using the `smbpasswd` command and its `-x` option, as in **`smbpasswd -x sally`** to delete the `sally` account from Samba's database.

Verifying Account Use

Once accounts are created, you may have cause to check who's using the computer. This information can be useful for a variety of reasons—to see how many people might be inconvenienced if you shut down the computer for maintenance or to verify that an account is *not* in use if the user is away on vacation, for instance. Several tools are useful in this task, including `who`, `w`, and `last`. The `whoami` command is also potentially useful if you regularly use multiple accounts.

Checking Who's Logged In

The `who` and `w` commands are similar in purpose. Both display information on who is currently logged in. For instance, consider the following interactions:

```
$ who
paul      :0          2009-04-08 17:26
paul      pts/0      2009-04-08 17:26 (:0.0)
rhonda    pts/5      2009-04-10 20:06 (tycho.luna.edu)
```

```
$ w
20:10:18 up 2 days,  2:45,  7 users,  load average: 0.00, 0.00, 0.00
USER      TTY      FROM            LOGIN@   IDLE   JCPU   PCPU WHAT
paul      :0        -                Wed17    ?xdm?  9:04   0.13s /usr/bin/gnome-
paul      pts/0     :0.0            Wed17    1:29m  0.35s  0.08s bash
rhonda    pts/5     tycho.luna.edu   20:06    0.00s  0.02s  0.01s w
```

Both of these interactions reveal three current logins: two by paul and one by rhonda. The user paul appears to be logged in at the console, as revealed by the :0 and :0.0 fields in the TTY and FROM columns of the w output and by the similar data in who's output. These identifiers refer to a local X server. The tycho.luna.edu field in rhonda's data, on the other hand, suggests that she's logged in over a network, from the named remote computer.

The default output for who is fairly basic, so it's a good tool if you just want a summary of who's logged on. Table 5.4 summarizes options to who that can increase or change the information it presents. Some of these options overlap with the function of other tools, such as runlevel and uptime.

TABLE 5.4 Options for who

Option Name	Option Abbreviation	Effect
--all	-a	Presents all the available information.
--boot	-b	Displays the system boot time instead of a list of logged-in users.
--dead	-d	Displays dead processes, which can be useful information when managing processes.
--heading	-H	Adds column headings, which are omitted by default.
--ips	None	Causes remote systems to be identified by IP address rather than hostname. Not all versions of who support this option.
--login	-l	Obtains a list of running system login processes instead of a list of logged-in users.
--lookup	None	Looks up remote systems' IP addresses to obtain hostnames.
None	-m	Restricts the username list to the user who's using standard input.
--process	-p	Lists processes spawned by init.

TABLE 5.4 Options for `who` (continued)

Option Name	Option Abbreviation	Effect
<code>--count</code>	<code>-q</code>	Obtains the number of users who are logged in.
<code>--runlevel</code>	<code>-r</code>	Displays the current runlevel (similar to the <code>runlevel</code> command).
<code>--short</code>	<code>-s</code>	Displays only the username, access method, and login time; it's the default.
<code>--time</code>	<code>-t</code>	Finds when the system clock last changed.
<code>--users</code>	<code>-u</code>	Displays users who are logged in; it's similar to the default, but it adds a few fields.

The `w` command presents more information by default than does `who`. Like `who`, `w` has a number of options, which are summarized in Table 5.5. You can also pass a username to `w`, as in `w paul`, to obtain information about only that user's logins.

TABLE 5.5 Options for `w`

Option Name	Effect
<code>-h</code>	Prints output without the header.
<code>-u</code>	Ignores the username when computing process and CPU times; has a subtle effect when using <code>su</code> , <code>sudo</code> , and similar tools.
<code>-s</code>	Use short format; omits several columns' data from the output.
<code>-f</code>	Toggles printing the FROM field; this field might or might not be included by default, depending on compile-time options.
<code>-o</code>	Use old-style display format. (Display format has changed slightly recently.)

Both `who` and `w` are useful tools when you need to check on the current status of a computer's user loads—say to minimize the inconvenience caused by shutting down a system for maintenance. The `w` command also gives you some idea of what your users are doing. This last information is very limited, though; `w` was designed long before GUIs became common, so it can't show you information on the many independent windows a user may have open. The `ps` command (described in Chapter 3) is more likely to be useful for finding such information.

Checking Historical Login Data

When you need to find out who’s logged into your computer in the recent past, neither who nor w will do the job. Instead, you need `last`, which displays a list of recent logins, most recent first:

```
$ last -6
rhonda pts/6      tycho.luna.edu  Fri Apr 10 20:59  still logged in
rhonda pts/5      tycho.luna.edu  Fri Apr 10 20:06  still logged in
paul   pts/4      :0.0           Fri Apr 10 15:53  still logged in
root   pts/4      tranquility.luna Fri Apr 10 09:53 - 09:56 (00:02)
paul   pts/4      tranquility.luna Fri Apr 10 08:15 - 08:46 (00:31)
paul   pts/5      tycho.luna.edu  Thu Apr 9 21:45 - 22:04 (00:18)
```

wtmp begins Tue Apr 7 06:20:29 2009

This output reveals that three logins are still active but that several others have been terminated. (In fact, the `-6` option limited the output; without that option, `last`’s output is likely to go on for pages!) Table 5.6 summarizes `last`’s options.

TABLE 5.6 Options for `last`

Option Name	Effect
<code>-f file</code>	Ordinarily, <code>last</code> gets its data from the <code>/var/log/wtmp</code> file. You can specify another file with this option, which can be useful if you’re studying login activity from an emergency boot system.
<code>-num</code>	You can specify a number (<i>num</i>) to limit output to that many logins.
<code>-t YYYYMMDDHHMMSS</code>	You can find out who was logged in at a particular time with this option, which requires a time specification down to the second.
<code>-R</code>	This option causes the hostname field to be omitted from the output.
<code>-a</code>	This option shifts the hostname to the final field.
<code>-d</code>	This option causes the IP address of remote systems to be looked up anew.
<code>-i</code>	You can obtain the IP address of a remote system, rather than its hostname, with this option.

TABLE 5.6 Options for `last` (continued)

Option Name	Effect
<code>-o</code>	You can read old-style <code>wtmp</code> files, created by <code>libc5</code> programs, with this option. (<code>libc5</code> was abandoned years ago, so this won't be useful unless you're running some very old software.)
<code>-x</code>	This option adds system runlevel changes, including shutdowns, to the output.

Verifying Your Identity

As a system administrator, you may occasionally need to use `su` to run programs using other users' identities in order to debug problems or to acquire root privileges. You might also have multiple login accounts, either on one system or on many. In any of these cases, you may forget what account you're using. When this happens, `whoami` can come to the rescue:

```
$ whoami
c.farrell
```

Few options to `whoami` exist; just `--help` and `--version`, which are common ones on many programs.

Configuring Groups

Linux provides group configuration tools that parallel those for user accounts in many ways. Groups are not accounts, however, so many features of these tools differ. Likewise, you can create or modify groups by directly editing the configuration files in question. Their layout is similar to that for account control files, but the details differ.

Adding Groups

Linux provides the `groupadd` command to add a new group. This utility is similar to `useradd` but has fewer options. The `groupadd` syntax is as follows:

```
groupadd [-g GID [-o]] [-r] [-f] groupname
```

The parameters to this command enable you to adjust its operation, as summarized in Table 5.7.

TABLE 5.7 Options for groupadd

Option Name	Option Abbreviation	Effect
--gid <i>GID</i>	-g	You can provide a specific GID with this parameter. If you omit it, groupadd uses the next available GID.
--non-unique	-o	Normally, the GID you specify must be unused by other groups, but the -o parameter overrides this behavior, enabling you to create multiple groups that share one GID.
--system	-r	This parameter instructs groupadd to create a group with a GID of less than 500. Not all distributions support this option; it was added by Red Hat, and has been used on some related distributions, because Red Hat uses GIDs of 500 and greater for user private groups (as described shortly, in the section “Using User Private Groups”).
--force	-f	Normally, if you try to create a group that already exists, groupadd returns an error message. This parameter suppresses that error message. Not all versions of groupadd support this parameter.

In most cases, you’ll create groups without specifying any parameters except for the group name itself, thus:

```
# groupadd project3
```

This command creates the `project3` group, giving it whatever GID the system finds convenient—usually the highest existing GID plus 1. Once you’ve done this, you can add users to the group, as described in the next section. When you add new users, you can add them directly to the new group with the `-g` and `-G` parameters to `useradd`, described earlier.

Modifying Group Information

You can modify group information, such as user account information, either by using utility programs or by directly editing the underlying configuration file, `/etc/group`. As with creation, there are fewer options for modifying groups than for modifying accounts, and the utilities and configuration files are similar. In fact, `usermod` is one of the tools that’s used to modify groups.

Using *groupmod* and *usermod*

The *groupmod* command modifies an existing group's settings. Its syntax is as follows:

```
groupmod [-g GID [-o]] [-n newgroupname] oldgroupname
```

The options to this command modify its operation. The *-g* and *-o* options work just like their counterparts in *groupadd*. The *-n* (or *--new-name*) option specifies a new group name, should you want to change it.

One of the most common group manipulations you'll perform, however, is not handled through *groupmod*; it's done with *usermod*. Specifically, *usermod* enables you to add a user to a group with its *--groups* (or *-G*) parameter. For instance, the following command sets *sally* to be a member of the *users*, *project1*, and *project4* groups, and it removes her from all other groups:

```
# usermod -G users,project1,project4 sally
```



Be sure to list all the user's current groups in addition to any groups to which you want to add the user. Omitting any of the user's current groups will remove the user from those groups. You can discover the groups to which a user currently belongs with the *groups* command, as in **groups sally**. To avoid accidentally omitting a group, many system administrators prefer to modify the */etc/group* file in a text editor or to use *gpasswd*. Both methods enable you to add users to groups without specifying a user's existing group memberships.

Using *gpasswd*

The *gpasswd* command is the group equivalent to *passwd*. The *gpasswd* command also enables you to modify other group features and to assign *group administrators*—users who may perform some group-related administrative functions for their groups. The basic syntax for this command is as follows:

```
gpasswd [-a user] [-d user] [-R] [-r] [-A user[,...]] [-M user[,...]] group
```

The options for this command modify its actions, as summarized in Table 5.8. If entered without any parameters except a group name, *gpasswd* changes the password for the group. Group passwords enable you to control temporary membership in a group, as granted by *newgrp*. Ordinarily, members of a group may use *newgrp* to change their current group membership (affecting the group of files they create). If a password is set, even those who aren't members of a group may become temporary group members; *newgrp* prompts for a password that, if entered correctly, gives the user temporary group membership.

TABLE 5.8 Options for `gpasswd`

Option Name	Effect
<code>-a user</code>	This option adds the specified user to the specified group.
<code>-d user</code>	This option deletes the specified user from the specified group.
<code>-R</code>	This option configures the group to not allow anybody to become a member through <code>newgrp</code> .
<code>-r</code>	You can remove a password from a group using this option.
<code>-A user[,...]</code>	The root user may use this parameter to specify group administrators. Group administrators may add and remove members from a group and change the group password. Using this parameter completely overwrites the list of administrators, so if you want to add an administrator to an existing set of group administrators, you must specify <i>all</i> of their usernames.
<code>-M user[,...]</code>	This option works like <code>-A</code> , but it also adds the specified user(s) to the list of group members.

Unfortunately, some of these features are not implemented correctly in all distributions. In particular, password entry by nongroup members sometimes does *not* give group membership—the system responds with an access denied error message. The `-R` option also sometimes doesn't work correctly—group members whose primary group membership is with another group may still use `newgrp` to set their primary group membership.

Directly Modifying Group Configuration Files

Group information is stored primarily in the `/etc/group` file. Like account configuration files, the `/etc/group` file is organized as a set of lines, one line per group. A typical line in this file resembles the following:

```
project1:x:501:sally,sam,ellen,george
```

Each field is separated from the others by a colon. The meanings of the four fields are as follows:

Group name The first field (`project1` in the preceding example) is the name of the group.

Password The second field (`x` in the preceding example) is the group password. Distributions that use shadow passwords typically place an `x` in this field; others place the encrypted password directly in this field.

GID The group ID number (in this example's case, 501) goes in this field.

User list The final field is a comma-separated list (with no spaces) of group members.

Users may also be members of a group based on the primary group specification of their entries in the `/etc/passwd` file. For instance, if user `george` had `project1` listed as his primary group, he need not be listed in the `project1` line in `/etc/group`. If user `george` uses `newgrp` to change to another group, though, he won't be able to change back to `project1` unless he's listed in the `project1` line in `/etc/group`.

Systems with shadow passwords also use another file, `/etc/gshadow`, to store shadow password information on groups. This file stores the shadow password and information on group administrators, as described earlier in “Using *gpasswd*.”



If you configure Linux to use a network account database, the `/etc/group` file will be present and may define groups important for the system's basic operation. As with `/etc/passwd` and `/etc/shadow`, though, important user groups are likely to be defined only on the network account server, not in `/etc/group`.

Deleting Groups

Deleting groups is done via the `groupdel` command, which takes a single argument: a group name. For instance, typing `groupdel project3` removes the `project3` group. You can also delete a group by editing the `/etc/group` file (and `/etc/gshadow`, if present) and removing the relevant line for the group. It's generally better to use `groupdel`, though, because `groupdel` checks to see whether the group is any user's primary group. If it is, `groupdel` refuses to remove the group; you must change the user's primary group or delete the user account first.

As with deleting users, deleting groups can leave “orphaned” files on the computer. You can locate them with the `find` command, which is described in more detail in Chapter 2. For instance, if the deleted group had used a GID of 503, you can find all the files on the computer with that GID by using the following command:

```
# find / -gid 503
```

Once you've found any files with the deleted group's ownership, you must decide what to do with them. In some cases, leaving them alone won't cause any immediate problems, but if the GID is ever reused, it could lead to confusion and even security breaches. Therefore, it's usually best to delete the files or assign them other group ownership using the `chown` or `chgrp` command.

Using Common User and Group Strategies

Linux's tools for handling users and groups can be quite powerful, but until you have some experience using them in a practical working environment, it's not always easy to see how best to use them. This is also one area of system configuration that can't be preconfigured

by distribution maintainers in a way that's very helpful. After all, user accounts and groups are necessarily local features—your system's users and groups will almost certainly be different from those of a system across town. Nonetheless, Linux distributions need to have some sort of default scheme for handling users and groups—what UIDs and GIDs to assign and what groups to use for newly created users by default. Two such schemes are in common use, and each can be expanded in ways that may be useful to your system.



The strategies described here can be further modified by employing access control lists (ACLs), as described in Chapter 2. These have the effect of giving individual users fine-grained control over who may access their files.

Using User Private Groups

The *user private group* scheme is used by Red Hat Linux and some of its derivative distributions, such as Fedora and Mandriva. This scheme creates an initial one-to-one mapping of users and groups. In this system, whenever a user account is created, a matching group is created, usually of the same name. This matching group has one member—its corresponding user. For instance, when you create the account `sally`, a group called `sally` is also created. The account `sally`'s primary group is the group `sally`. When used without modification, the user private group strategy effectively eliminates groups as a factor in Linux security—because each group has just one member, group permissions on files become unimportant.

It's possible to modify group membership to control file access, however. For instance, if you want the user `george` to be able to read `sally`'s files, you can add `george` to the `sally` group and set the `sally` user's `umask` to provide group read access to new files created by the user `sally`. Indeed, if you make all users group administrators of their own groups, users may control who has access to their own files by using `gpasswd` themselves. Overall, this approach provides considerable flexibility, particularly when users are sophisticated enough to handle `gpasswd`. Giving users such power may run counter to your system's security needs, though. Even when security policy dictates against making users group administrators, a user private group strategy can make sense if you need to fine-tune file access on a user-by-user basis. This approach can also provide asymmetrical file access. For instance, `george` may be able to read `sally`'s files (at least, those with appropriate group permissions), but `sally` might not have access to `george`'s files (unless `george` sets the world read bit on his files).

Using Project Groups

A second approach to group configuration is to create separate groups for specific purposes or projects. Therefore, I refer to this as the project group approach. Consider an example of a company that's engaged in producing three different products. Most employees work

on just one product, and for security reasons, you don't want users working on one product to have access to information relating to the other two products. In such an environment, a Linux system may be well served by having three main user groups, one for each product. Most users will be members of precisely one group. If you configure the system with a umask that denies world access, those who don't belong to a specific product's group won't be able to read files relating to that product. You can set read or read/write group permission to allow group members to easily exchange files. (Individual users may use `chmod` to customize permissions on particular files and directories, of course.) If a user needs access to files associated with multiple products, you can assign that user to as many groups as necessary to accommodate the need. For instance, a supervisor might have access to all three groups.

The project group approach tends to work well when a system's users can be easily broken down into discrete groups whose members must collaborate closely. It can also work well when users need not (and even should not) have ready access to each other's files, as with students taking the same class. In such a case, you would set the umask to block group access to users' files. The logical grouping of users can still be helpful to you as an administrator, however, because you can track users according to their group—you can easily find all files owned by users taking a particular class, for instance. (Keep in mind that this tracking ability breaks down when users are members of multiple groups.)

Many Linux distributions default to using a type of project group approach. The default primary group for new users on such systems is typically called `users`. You can, and in most cases should, create additional groups to handle all your projects. You can leave the `users` group intact but not use it, rename it to the first project group name, or use `users` as an overarching group for when you want to give access to a file to most ordinary users, but perhaps not everyone (such as guest users on an FTP server).

Assigning Users to Multiple Groups

On any but very simple systems, it's likely that at least some users will be members of multiple groups. This means that these users will be able to do the following things:

- Read files belonging to any of the user's groups, provided that the file has group read permission.
- Write to existing files belonging to any of the user's groups, provided that the file has group write permission.
- Run programs belonging to any of the user's groups, provided that the file has group execute permission.
- Change the group ownership of any of the user's own files to any of the groups to which the user belongs.
- Use `newgrp` to make any of the groups to which the user belongs the user's primary group. Files created thereafter will have the selected group as the group owner.

Multiple group membership is extremely important when using user private groups, as described earlier—without this, it's impossible to fine-tune access to users' files. Even in a project group configuration, though, multiple group membership is critical for users who need access to multiple groups' files.

You may find yourself creating a group membership scheme that's some combination of these two, or one that's unique unto itself. For instance, you might create multiple overlapping subgroups in order to fine-tune access control. It might be common in such a situation for users to belong to multiple groups. Part of the problem with such configurations is in teaching users to properly use the `newgrp` and `chgrp` commands—many less technically savvy users prefer to simply create files and not worry about such details.

Improving Account Security

Creating, maintaining, and removing user accounts are obviously important activities on a Linux system. One particularly essential account maintenance task (or set of tasks) is maintaining account security. Crackers sometimes attack a system through vulnerable user accounts. Once access to a normal account is achieved, bugs or lax internal security can be exploited to enable the attacker to acquire root access, or the account can be used to attack other systems. Therefore, it's vital that you attend to this matter by periodically reviewing your configuration to see that it remains secure.



The popular media uses the term *hacker* to refer to computer miscreants. This term has an older meaning, however—somebody who enjoys programming or doing other technically challenging work on computers, but not in an illegal or destructive sense. Many Linux programmers consider themselves hackers in this positive sense. Thus, I use the term *cracker* to refer to those who break into computers.

Enforcing User Password Security

As a general rule, people tend to be lazy when it comes to security. In computer terms, this means that users tend to pick passwords that are easy to guess, and they change them infrequently. Both these conditions make a cracker's life easier, particularly if the cracker knows the victim. Fortunately, Linux includes tools to help make your users select good passwords and change them regularly.

Common (and therefore poor) passwords include those based on the names of family members, friends, and pets; favorite books, movies, television shows, or the characters in any of these; telephone numbers, street addresses, or Social Security numbers; or other meaningful personal information. Any single word that's found in a dictionary (in *any* language) is a poor choice for a password. The best possible passwords are random collections of letters,

digits, and punctuation. Unfortunately, such passwords are difficult to remember. A reasonable compromise is to build a password in two steps. First, choose a base that's easy to remember but difficult to guess. Second, modify that base in ways that increase the difficulty of guessing the password.

One approach to building a base is to use two unrelated words, such as *bun* and *pen*. You can then merge these two words (*bunpen*). Another approach, and one that's arguably better than the first, is to use the first letters of a phrase that's meaningful to the user. For instance, the first letters of "yesterday I went to the dentist" become *yiwtttd*. In both cases, the base should not be a word in any language. As a general rule, the longer the password, the better. Older versions of Linux had password length limits of eight characters, but those limits have been lifted by the use of the MD5 and SHA password hashes, which are standard on modern Linux distributions. Many Linux systems require passwords to be at least four to six characters in length; the `passwd` utility won't accept anything shorter than the distribution's minimum.

With the base in hand, it's time to modify it to create a password. The user should apply at least a couple of several possible modifications:

Adding numbers or punctuation The single most important modification is to insert random numbers or punctuation in the base. This step might yield, for instance, *bu3npe&n* or *y#i9wttd*. As a general rule, add at least two symbols or numbers.

Mixing case Linux uses case-sensitive passwords, so jumbling the case of letters can improve security. Applying this rule might produce *Bu3nPE&n* and *y#i9WttD*, for instance.

Order reversal A change that's very weak by itself but that can add somewhat to security when used in conjunction with the others is to reverse the order of some or all letters. You might apply this to just one word of a two-word base. This could yield *nu3BPE&n* and *DttW9i#y*, for instance.

Your best tool for getting users to pick good passwords is to educate them. Tell them that passwords can be guessed by malicious individuals who know them, or even who target them and look up personal information in telephone books, on Web pages, and so on. Tell them that, although Linux encrypts its passwords internally, programs exist that feed entire dictionaries through Linux's password encryption algorithms for comparison to encrypted passwords. If a match is found, the cracker has found the password. Therefore, using a password that's not in a dictionary, and that isn't a simple variant of a dictionary word, improves security substantially. Tell your users that their accounts might be used as a first step toward compromising the entire computer, or as a launching point for attacks on other computers. Explain to your users that they should *never* reveal their passwords to others, even people claiming to be system administrators—this is a common scam, but real system administrators don't need users' passwords. You should also warn them not to use the same password on multiple systems because doing so quickly turns a compromised account on one system into a compromised account on all the systems. Telling your users these things will help them understand the reasons for your concern, and it is likely to help motivate at least some of them to pick good passwords.

If your users are unconcerned after being told these things (and in any large installation, some will be), you'll have to rely on the checks possible in `passwd`. Most distributions' implementations of this utility require a minimum password length (typically four to six characters). They also usually check the password against a dictionary, thus weeding out some of the absolute worst passwords. Some require that a password contain at least one or two digits or punctuation.



Password-cracking programs, such as Crack (<http://www.crypticide.com/a1ecm/security/crack/>), are easy to obtain. You might consider running such programs on your own encrypted password database to spot poor passwords, and in fact, this is a good policy in many cases. It's also grounds for dismissal in many organizations and can even result in criminal charges being brought, at least if done without authorization. If you want to weed out bad passwords in this way, discuss the matter with your superiors and obtain written permission from a person with the authority to grant it before proceeding. Take extreme care with the files involved, too; it's probably best to crack the passwords on a computer with *no* network connections.

Another password security issue is password changes. Changing passwords on a frequent basis minimizes the window of opportunity for crackers to do damage; if a cracker obtains a password but it changes before the cracker can use it (or before the cracker can do further damage using the compromised account), the password change has averted disaster. As described earlier in this chapter, you can configure accounts to require periodic password changes. When so configured, an account will stop accepting logins after a time if the password isn't changed periodically. (You can configure the system to warn users when this time is approaching.) This is a very good option to enable on sensitive systems or those with many users. Don't set the expire time too low, though—if users have to change their passwords too frequently, they'll probably just switch between a couple of passwords, or they'll pick poor ones. Precisely what “too low” a password change time is depends on the environment. For most systems, 1–4 months is probably a reasonable change time, but for some it might be longer or shorter.

Steps for Reducing the Risk of Compromised Passwords

Passwords can end up in crackers' hands in various ways, and you must take steps to minimize these risks. Steps you can take to improve your system's security include the following:

Use strong passwords. Users should employ good passwords, as just described. This practice won't eliminate all risk, though.

Change passwords frequently. As just mentioned, changing passwords frequently can minimize the chance of damage due to a compromised password.

Use shadow passwords. If a cracker who's broken into your system through an ordinary user account can read the password file or if one of your regular users is a cracker who has

access to the password file, that individual can run any of several password-cracking programs on the file. For this reason, you should use shadow passwords stored in `/etc/shadow` whenever possible. Most Linux distributions use shadow passwords by default.

Keep passwords secret. You should remind your users not to reveal their passwords to others. Such trust is sometimes misplaced, and sometimes even a well-intentioned password recipient might slip up and let the password fall into the wrong hands. This can happen by writing the password down, storing it in electronic form, or sending it by e-mail or other electronic means. Indeed, users shouldn't e-mail their own passwords even to themselves, because e-mail can be intercepted.

Use secure remote login protocols. Certain remote login protocols are inherently insecure; all data traverse the network in an unencrypted form. Intervening computers can be configured to snatch passwords from such sessions. Because of this, it's best to disable Telnet, FTP, and other protocols that use cleartext passwords, in favor of protocols that encrypt passwords, such as SSH. Chapter 9, "Configuring Advanced Networking," and Chapter 11, "Configuring Network Servers II," describe these protocols in more detail.

Be alert to shoulder surfing. If your users work on computers in public, as is common in college computer centers, in Internet cafes, when using a laptop in public, and so on, it's possible that others will be able to watch them type their passwords. This practice is sometimes called *shoulder surfing*. Users should be alert to this possibility and minimize such logins if possible.

Some of these steps are things you can do, such as replacing insecure remote login protocols with encrypted ones. Others are things your users must do. Once again, this illustrates the importance of user education, particularly on systems with many users.

Disabling Unused Accounts

Linux computers sometimes accumulate unused accounts. This occurs when employees leave a company, when students graduate, and so on. You should be diligent about disabling such accounts because they can easily be abused, either by the individual who's left your organization or by others who discover the password through any of the means already described. As covered in detail earlier in this chapter, you disable accounts with the `userdel` command.

If the individual has had legitimate access to the `root` account, you must carefully consider how to proceed. If you have no reason to suspect the individual of wrongdoing, changing the root password and deleting the user's regular account are probably sufficient. If the individual might have sabotaged the computer, though, you'll have a hard time checking for every possible type of damage, particularly if the individual was a competent administrator. In such situations, you're better off backing up user data and reinstalling from scratch, just as you should if your system is compromised by an outsider.

Many Linux distributions create a number of specialized accounts that are normally not used for conventional user logins. These may include `daemon`, `lp`, `shutdown`, `mail`, `news`, `uucp`, `games`, `nobody`, and others. Some of these accounts have necessary functions. For instance, `daemon` is used by some servers, `lp` is associated with Linux's printing system, and `nobody` is

used by various programs that don't need high-privilege access to the system. Other accounts are likely to be unused on many systems. For instance, `games` is used by some games, and so it isn't of much use on most servers or true productivity workstations. You may be able to delete some of these unused specialty accounts, but if you do so, you should definitely record details on the accounts' configurations so that you can restore them if you run into problems because the account wasn't quite as unnecessary as it first seemed.

Using Shadow Passwords

Most Linux distributions use shadow passwords by default, and for the most part, this chapter is written with the assumption that this feature is active. In addition to providing extra security by moving hashed passwords out of the world-readable `/etc/passwd` file and into the more secure `/etc/shadow` file, shadow passwords add extra account information. (The earlier section “Directly Modifying Account Configuration Files” describes the `/etc/shadow` file's contents in detail.)



If you happen to be using a system that hasn't enabled shadow passwords but you want to add that support, you can do so. Specifically, the `pwconv` and `grpconv` programs do this job for `/etc/passwd` and `/etc/group`, respectively. These programs take no parameters; simply type their names to create `/etc/shadow` and `/etc/gshadow` files that hold the passwords and related account control information, based on the contents of the unprotected files.

One of the advantages of shadow passwords is that the Linux shadow password system enables use of more advanced password hashes. The earliest Linux systems used a *Triple Data Encryption Standard (3DES)* hash. This hash, although good not too many years ago, is outdated by today's standards. Many Linux distributions today use the *Message Digest 5 (MD5)* hash instead, but some have moved on even from that to use the *Secure Hash Algorithm (SHA)*, which is even more secure. Linux's password tools support passwords longer than eight characters for MD5 or SHA hashes, but not for 3DES hashes. You can tell which one your distribution uses by examining the password field in `/etc/shadow` (or `/etc/passwd`, if you're not using shadow passwords). MD5 passwords begin with the string `1`, SHA passwords begin with `5` or `6`, and 3DES passwords don't begin with any of these strings.

You can't convert existing passwords from one hash to another, except by reconfiguring your system to use the new password and then having users reenter their passwords. You can, though, tell Linux to use one tool or the other for new passwords. To do so, edit the `/etc/pam.d/passwd` file. This file controls the Pluggable Authentication Modules (PAM) configuration for the `passwd` utility—that is, it controls how `passwd` does its work. Locate a line that looks something like this:

```
password    required    pam_unix.so    nullok use_authok
```



The line may look somewhat different on your system, but the key points are that it begins with the string `password` and references the `pam_unix.so` library file. PAM is described in more detail in Chapter 12.

To configure Linux to encode new passwords using MD5, add `md5` to the list of parameters after `pam_unix.so`. To switch to SHA, add `sha256` or `sha512` to the parameter list (the number refers to the number of bits used in the hash, with higher values being preferable).

If you can't find a line that looks like this one in `/etc/pam.d/passwd`, look for it in `/etc/pam.d/system-auth`. This file serves as a stand-in for several other configuration files in some distributions, so making the change there may do the job.



Additional encryption and hashing functions exist, such as *MD4* and *blowfish*. These are not commonly used in Linux password databases, but they may be used in network account database systems, in encrypted data transfer tools such as SSH, and elsewhere.

Controlling System Access

One of the many uses of accounts is as a tool to control access to a computer—that is, to allow `sally` to log in remotely, but not `sam`, much less a cracker from halfway around the world. Precisely how such tasks can be accomplished varies from one server program to another; some provide better access controls than others, and each server has its own methods of handling the matter. The `root` account is often handled in a unique way with respect to access control, and for good reason—`root` is powerful enough that you may want to restrict its access to local logins in order to minimize the chance of abuse by a distant cracker.

Accessing Common Servers

Several common server programs provide some means to limit access by username, often via PAM. Others don't provide this facility but do provide a way to limit access by computer hostname or IP address. Servers you're particularly likely to want to limit in these ways include login servers and the File Transfer Protocol (FTP) server.



All servers have their own security implications. Chapter 10 describes some of these servers in brief, but you should be aware that running *any* server has security consequences that can be subtle, making safe configuration difficult. You should consult appropriate server-specific or security documentation before running servers on the Internet.

Controlling Login Access

Remote login access is usually provided by a Telnet or SSH server. These servers provide remote text-mode access, enabling users to run Bash or other shells and text-mode programs. SSH also supports tunneling X connections, so if the user's computer runs an X server, the user can run X programs hosted on the login server computer.

Unfortunately, Telnet provides only very limited security options. The server uses the `login` program to process user logins, so user-by-user login restrictions are those provided by `login`.

Telnet servers are usually called from a super server (`inetd` or `xinetd`), so you can use these servers' features (or the features of programs they call, such as TCP Wrappers) to restrict access to Telnet on the basis of the calling systems' IP addresses. These options are described in Chapter 4, "Managing System Services."

SSH servers don't normally use `login` to control the login process. These servers may employ PAM, though, so you can configure PAM to limit who may log in via SSH.

In addition to these controls, other options can limit `root` access to these login servers, as described later in "Controlling *root* Access."

Controlling FTP Access

Most FTP servers use PAM to authenticate user access. Instead of or in addition to PAM, some FTP servers use a file called `/etc/ftpusers` to limit who may access the computer. This file contains a list of users who may *not* access the FTP server. Typically, it includes `root` and various other system accounts, but you can add ordinary users to the list, if you like. You cannot use this file to restrict access based on the remote system, though, unlike the PAM-based restrictions.



The presence of an `/etc/ftpusers` file doesn't mean that the file is actually in use. This file could be a relic from an earlier FTP server installed on the system, or the FTP server might be configured to not use the file. Before you rely on this file, you should test it by adding an ordinary username to the file and then attempting to access your FTP server using that username. If you succeed, try restarting the FTP server and testing it again. If you can still access the FTP server, consult its documentation to learn how to get it to use `/etc/ftpusers` or use PAM to limit access.

Controlling *root* Access

The `root` account is unusually powerful, so a compromise of that account is far more serious than a compromise of an ordinary account. Furthermore, in many cases, `root` should not be accessing a computer over a network, at least not in certain ways. For instance, a standard Telnet server doesn't encrypt its traffic, so its use for `root` access exposes the `root` password on the network wires—or over radio waves, if you're using wireless networking. What's more, the

data transferred by root may be unusually sensitive. For instance, you might edit `/etc/shadow` as root, thus exposing the data in that file, if you edit it via a Telnet link.

For these reasons, many servers and login protocols provide extra tools to help control root access to the system. Frequently, these tools simply deny all access to direct root logins. Administrators can still log in using a normal account and then use `su`, `sudo`, or similar tools to perform administrative tasks. This approach requires two passwords, which means that a miscreant is less likely to be able to get in than if direct root logins were accepted.

The default configuration for most Linux distributions is to deny direct root logins via Telnet (or any other remote login server that uses the `login` program). Thus, you shouldn't need to change anything to keep this server from accepting root logins. If you do, the key lies in the `/etc/securetty` file. This file holds a list of terminals from which root is permitted to log in. It normally contains a list of local device filenames, one per line, minus the leading `/dev` directory indicator. If this list is incomplete, you may not be able to log in as root from the console. Adding appropriate specifications, such as `tty1` through `tty6` and `vc/1` through `vc/6`, should fix the problem. If you want to use a directly connected “dumb” RS-232 serial terminal, you can add its device filename, such as `ttyS0`. (You'll also need to enable this terminal for normal logins by adding it to the `/etc/inittab` file or creating a `ttyS0` file with appropriate options in `/etc/event.d`, depending on the version of `init` you're using.)



Even if you log in using Telnet via a normal user account, using `su` and performing administrative functions can be risky. The password you type after you type `su` will be passed over the network in an unencrypted form, as will all the data you type or see on your screen. Remote text-mode administration is best done via SSH or some other encrypted protocol.

Most default SSH configurations allow root to log in directly. Although SSH's encryption makes this practice much safer than the equivalent when using Telnet, you can gain the added benefit of requiring two passwords by disabling direct root logins via SSH. To do so, you must edit the server's `/etc/ssh/sshd_config` file (not to be confused with `ssh_config`, which controls the SSH client). Look for the `PermitRootLogin` line, and set it to `no`:

```
PermitRootLogin no
```

You may want to consult the documentation for other servers you run, as well. Some, including remote administration tools such as the Samba Web Administration Tool (SWAT), require root access to do more than display basic information and perhaps change user passwords. Others, such as the main Samba servers themselves, should ordinarily not give root access—they simply aren't designed for administrative functions, and root may be able to do things with the server that you'd rather not be done. For the most part, remote root access should be limited to SSH (ideally *after* a regular user login) or to tools that are explicitly designed to support root access for administrative purposes.

Summary

Linux's accounts and its security model are inextricably intertwined. A single Linux system can support many users, who can be tied together in groups. Users can create files that they own and that have permissions that define which other users may access the files and in what ways. To manage these users, you'll use a handful of commands, such as `useradd`, `groupadd`, `userdel`, `usermod`, and `passwd`. These commands enable you to manage your user accounts to suit your system's needs.

Managing account security is critically important. You must educate your users about the importance of good passwords, and about proper procedures for safeguarding their passwords. Most importantly, users should know to *never* divulge their passwords to others. They should also be alert to suspicious activities that might indicate shoulder surfing or other methods crackers employ to obtain passwords. As a system administrator, you can disable or delete unused accounts and manage shadow passwords.

Controlling access to a computer is an important part of security and user management. Many programs also provide their own tools to accomplish these goals. Some servers provide special options to disable or limit root access to the computer, and you should often take advantage of such options.

Exam Essentials

Describe why accounts are important on a Linux system. Accounts enable several users to work on a computer with minimal risk that they'll damage or (if you so desire) read one another's files. Accounts also enable you to control normal users' access to critical system resources, limiting the risk of damage to the Linux installation as a whole.

Summarize important files in controlling access to Linux. The `/etc/passwd` and `/etc/shadow` files contain information on Linux accounts. Files in `/etc/pam.d` control PAM, including defining how PAM authenticates users. Individual programs and servers often have their own security files, such as `/etc/sudoers` to control `sudo`, `/etc/ftpusers` to control who may access an FTP server, and `/etc/ssh/sshd_config` to control the SSH server.

Describe the characteristics of a good password. Good passwords resemble random strings of letters, numbers, and punctuation. To make them memorable to the account holder, they can be generated by starting from a base built on a personally relevant acronym or a pair of unrelated words and then modified by adding letters and punctuation, mixing the case of letters, and reversing some sequences in the password.

Explain the importance of shadow passwords. The shadow password system stores password hashes in a file that can be read only by root, thus reducing the risk that a cracker can read the file and use a password-cracking program to discover users' passwords.

Summarize Linux password hashes. Linux has traditionally used 3DES, but modern distributions use the MD5 or SHA hash instead. Other password hashes are possible and are sometimes used with network authentication tools.

Describe methods of deleting user accounts. Accounts can be deleted by deleting the appropriate entries in `/etc/passwd` and `/etc/shadow` or by using utilities such as `userdel`. You might also need to delete user files (`userdel` can optionally do at least part of this job) and delete or change references to the user in other configuration files, such as `/etc/samba/smbpasswd`.

Summarize why using root access is dangerous. Every time the root password is entered is a chance for it to be discovered, so overusing the root account increases the odds that your computer will be compromised. Commands can also contain typos or other errors, and when this happens as root, the consequences can be far more damaging than is the case when an ordinary user mistypes a command.

Review Questions

1. Which of the following are legal Linux usernames? (Choose all that apply.)
 - A. Larrythemoose
 - B. 4sale
 - C. PamJones
 - D. Samuel_Bernard_DeLaney_the_Fourth
2. Why are groups important to the Linux user administration and security models?
 - A. They can be used to provide a set of users with access to files, without giving *all* users access to the files.
 - B. They allow you to set a single login password for all users within a defined group.
 - C. Users may assign file ownership to a group, thereby hiding their own creation of the file.
 - D. By deleting a group, you can quickly remove the accounts for all users in the group.
3. Which of the following actions allow one to perform administrative tasks? (Choose all that apply.)
 - A. Logging in as an ordinary user and using the `chgrp` command to acquire superuser privileges
 - B. Logging in at the console with the username `root`
 - C. Logging in as an ordinary user and using the `su` command to acquire superuser privileges
 - D. Logging in when nobody else is using the system, thus using it as a single-user computer
4. How may you limit the number of prior logins returned by the `last` command?
 - A. `-i`, to limit output to important (`root` and other administrative) logins
 - B. `-since YYYYMMDD`, where `YYYYMMDD` is a date
 - C. `-limit num`, where `num` is a number
 - D. `-num`, where `num` is a number
5. Which of the following is true of Linux passwords?
 - A. They are changed with the `password` utility.
 - B. They may consist of lowercase letters and numbers only.
 - C. They must be changed once a month.
 - D. They may be changed by the user who owns an account or by `root`.
6. Which of the following commands configures the `laura` account to expire on January 1, 2005?
 - A. `chage -I 2005-01-01 laura`
 - B. `userchange -e 2005-01-01 laura`
 - C. `usermod -e 2005 laura`
 - D. `chage -E 2005-01-01 laura`

7. Which of the following does **groupadd** allow you to create?
 - A. One group at a time
 - B. An arbitrary number of groups with one call to the program
 - C. Only user private groups
 - D. Passwords for groups
8. Which of the following is true of the **groupdel** command? (Choose all that apply.)
 - A. It won't remove a group if that group is any user's default group.
 - B. It won't remove a group if the system contains any files belonging to that group.
 - C. It removes references to the named group in **/etc/group** and **/etc/gshadow**.
 - D. It won't remove a group if it contains any members.
9. Which of the following describes the user private group strategy?
 - A. It is a requirement of the Red Hat and Mandriva distributions.
 - B. It cannot be used with Debian GNU/Linux.
 - C. It lets users define groups independently of the system administrator.
 - D. It creates one group per user of the system.
10. Which of the following is true when a user belongs to the **project1** and **project2** groups?
 - A. The user must type **newgrp project2** to read files belonging to **project2** group members.
 - B. If group read permissions are granted, any file created by the user will automatically be readable to both **project1** and **project2** group members.
 - C. The user may use the **newgrp** command to change the default group associated with files the user subsequently creates.
 - D. The user's group association (**project1** or **project2**) just after login is assigned randomly.
11. How should you engage users in helping to secure your computer's passwords?
 - A. Educate them about the importance of security, the means of choosing good passwords, and the ways crackers can obtain passwords.
 - B. Give some of your users copies of the encrypted database file as backup in case a cracker breaks in and corrupts the original.
 - C. Enforce password change rules, but don't tell users how crackers obtain passwords since you could be educating a future cracker.
 - D. Instruct your users to e-mail copies of their passwords to themselves on other systems so that they're readily available in case of an emergency.
12. Which of the following accounts is the most likely prospect for deletion on a mail server?
 - A. **daemon**
 - B. **games**
 - C. **mail**
 - D. **nobody**

13. While looking at the `/etc/passwd` file, you notice that an `x` appears in the second field for every user. What does this indicate?
 - A. All passwords are set to expire.
 - B. Passwords do not expire.
 - C. Passwords are not required on the system.
 - D. Passwords are stored in the shadow file.
14. Which of the following utilities is used to convert conventional passwords to shadow passwords?
 - A. `skel`
 - B. `shadow`
 - C. `pwconv`
 - D. `crypt`
15. Which commands can be used to discover which of a system's users are currently logged in? (Choose all that apply.)
 - A. `login`
 - B. `who`
 - C. `which`
 - D. `w`
16. In performing your administrative duties, you've made heavy use of the `su` command to temporarily acquire various users' identities, and you've forgotten with which account a shell is currently associated. How might you resolve this question?
 - A. Type `whoami`.
 - B. Type `cat /etc/passwd`.
 - C. Select File ➤ User from the desktop environment's menu.
 - D. Type `bash` to start a new shell.
17. You have recently been assigned administration of an older Linux server using 3DES password encryption. You want to send an e-mail to users encouraging them to change their passwords. How long can their passwords be?
 - A. Eight characters
 - B. Fifteen characters
 - C. Thirty-two characters
 - D. Unlimited

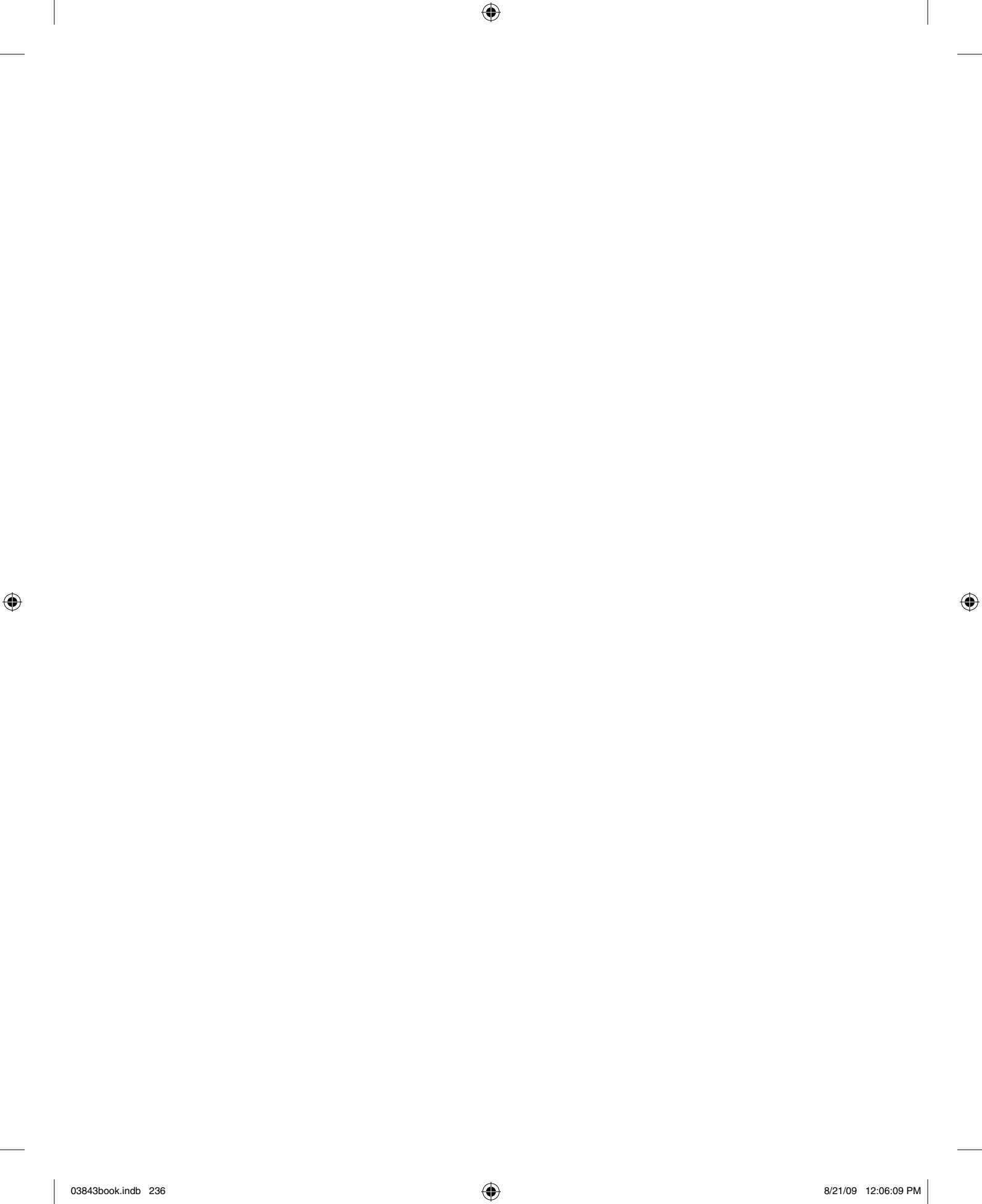
18. You are trying to explain to management why developers should use Linux to debug their applications. Which feature of Linux supports multiple simultaneous logins through the standard console and could be useful in application development?
- A. Multitasking
 - B. Multithreading
 - C. Virtual terminals
 - D. Concurrency
19. Which of the following commands can be used to delete a user named `kristin` and remove all files from the user's home directory, as well as the home directory itself?
- A. `userdel kristin`
 - B. `userdel -r kristin`
 - C. `userdel -a kristin`
 - D. `deluser -a kristin`
20. Which is the safest method of performing administrative tasks, from a security point of view?
- A. Logging in directly at the console as `root`
 - B. Using `sudo` after a console login as an ordinary user
 - C. Using `su` after a console login as an ordinary user
 - D. Logging in via Telnet as `root`

Answers to Review Questions

1. A, C. A Linux username must contain fewer than 32 characters and start with a letter, and it may consist of letters, numbers, and certain symbols. Options A and C both meet these criteria. (Option C uses mixed upper- and lowercase characters, which is legal but discouraged.) Option B begins with a number, which is invalid. Option D is longer than 32 characters.
2. A. Groups provide a good method of file-access control. Although they may have passwords, these are *not* account login passwords; those passwords are set on a per-account basis. Files do have associated groups, but these are *in addition* to individual file ownership, so they cannot be used to mask the file's owner. Deleting a group *does not* delete all the accounts associated with the group.
3. B, C. Direct login as `root` and using `su` to acquire `root` privileges from an ordinary login both allow a user to administer a system. The `chgrp` command is used to change group ownership of a file, not to acquire administrative privileges. Although Linux does support a single-user emergency rescue mode, this mode isn't invoked simply by having only one user logged on.
4. D. The `-num` parameter restricts `last` to the last *num* logins, as option D specifies. The `-i` parameter of option A causes output to show IP numbers rather than hostnames; it doesn't change the number of logins shown. The `last` command doesn't have `-since` or `-limit` parameters.
5. D. Both the superuser and the account owner may change an account's password. The utility for doing this is called `passwd`, not `password`. Although an individual user might use just lowercase letters and numbers for a password, Linux also supports uppercase letters and punctuation. The system administrator may enforce once-a-month password changes, but such changes aren't required by Linux per se.
6. D. Either `chage -E` or `usermod -e` may be used for this task, followed by a date expressed in `YYYY/MM/DD` or `YYYY-MM-DD` format. Option A uses the wrong parameter (`-I`), and option B uses the nonexistent `userchange` command. Option C is actually a legal command, but it specifies a date 2005 days after January 1, 1970—in other words, in mid-1975.
7. A. The `groupadd` command creates one group per call to the program. Such a group *may* be a user private group, but need not be. Group passwords are created with `gpasswd`, not `groupadd`.
8. A, C. The `groupdel` command modifies the group configuration files, and it checks the user configuration files to be sure that it doesn't "orphan" any users first. The group may contain members, though, as long as none lists the group as its primary group. The `groupdel` command performs no search for files belonging to the group, but it's a good idea for you to do this manually either before or after removing the group.

9. D. Although Red Hat and Mandriva use the user private group strategy by default, you can design and use another strategy. Likewise, you *may* use the user private group strategy with any Linux distribution, even if it doesn't use this strategy by default. Ordinary users can't create groups by themselves, although if they're group administrators in a user private group system, they may add other users to their own existing groups.
10. C. The `newgrp` command changes the user's active group membership, which determines the group associated with any files the user creates. This command is *not* required to give the user access to files with other group associations if the user is a member of the other group and the file has appropriate group access permissions. Files have exactly one group association, so a user who belongs to multiple groups must specify to which group any created files belong. This is handled at login by setting a default or primary group recorded with the user's other account defaults in `/etc/passwd`.
11. A. Education helps users to understand the reasons to be concerned, which can motivate conformance with password procedures. Cracking procedures are common knowledge, so withholding general information won't keep that information out of the hands of those who want it. Copying password files and sending unencrypted passwords through e-mail are both invitations to disaster; encrypted files can be cracked, and e-mail can be intercepted.
12. B. One or both of `daemon` and `mail` might be required by the mail server or other system software, so these are poor prospects for removal. Likewise, `nobody` is used by a variety of processes that need only low-privilege access rights. The `games` account is most frequently used by games for high-score files and the like and so is most likely unused on a mail server.
13. D. When an `x` appears for entries in the second field of the `passwd` file, it indicates that the passwords are stored elsewhere—in the `/etc/shadow` file. Expiration information is stored in `/etc/shadow`, not `/etc/passwd`. An account that does not require a password for login has an empty password field in `/etc/passwd` or `/etc/shadow`.
14. C. The `pwconv` utility is used to convert conventional passwords to shadow passwords (the opposite of this action is performed by `pwunconv`). `skel` is a file, not a utility, that holds a “skeleton” of settings to be applied to newly created users. The shadow file (`/etc/shadow`) is where the passwords are stored, but it is not a utility. `crypt` is a utility that hashes data; it can be used to encrypt passwords, but doesn't convert conventional to shadow passwords or vice versa.
15. B, D. The `who` and `w` commands both display lists of currently logged-in users, as the question specifies. The `login` program manages text-mode console logins and some types of remote logins; it presents the `login:` and `password:` prompts and then, if the user is authenticated, launches a shell. The `which` command tells you whether a command is internal to the shell, a shell alias, or an external command; it has nothing to do with who's logged in.
16. A. The `whoami` command displays the effective user ID—the username associated with the command, which will in turn be the username associated with the current shell. Option B will display the current account database file, but this information won't help answer the question of what account you're using. Even if a desktop environment has a File > User menu item, that item won't reliably tell you whose account you're using at a command shell. Any shell you launch from the current one will run with the current shell's privileges, so option D won't be effective.

17. A. Linux's password tools support passwords longer than eight characters for MD5 or SHA hashes, but not for 3DES hashes. 3DES hashes are limited to passwords of eight characters or less.
18. C. Linux supports multiple simultaneous logins through its standard console through the use of *virtual terminals* (VTs). From a text-mode login, pressing the Alt key along with a function key from F1 to F6 typically switches to a different virtual screen. Multitasking allows the machine to do more than one task at a time, while multithreading simply means that more than one thread can be executed at a time. Concurrency is not a common term used other than describing how many different users can log on at one time.
19. B. While the `userdel` utility removes the user, the `-r` parameter causes the system to remove all files from the user's home directory, as well as the home directory itself. There is no `-a` option for the `userdel` utility, and there is no standard utility in Linux named `deluser`.
20. B. The `sudo` program restricts `root` access to a single command at a time, thus minimizing the risk of mistakenly running a command as `root` when an ordinary user account would suffice. Of the options listed, options A and C are both somewhat less secure than using `sudo`, while option D is downright dangerous because of the risk of data interception.



Chapter 6

Managing Disks

THE FOLLOWING COMPTIA OBJECTIVES ARE COVERED IN THIS CHAPTER:

- ✓ **1.2 Implement partitioning schemes and filesystem layout using the following tools and practices (LVM, RAID, fdisk, parted, mkfs).**
- ✓ **1.3 Explain the purpose for using each of the following filesystem types (Local: EXT2, EXT3, Reiser, FAT, NTFS, VFAT, ISO9660. Network: NFS, SMBFS/CIFS).**
- ✓ **1.4 Conduct routine mount and unmount of filesystems (mount, umount, /etc/fstab).**
- ✓ **1.5 Explain the advantages of having a separate partition or volume for any of the following directories (/boot, /home, /tmp, /usr, /var, /opt).**
- ✓ **1.6 Explain the purpose of the following directories (/ , /bin, /dev, /etc, /mnt, /proc, /root, /sbin, /usr/bin, /usr/local, /usr/lib, /usr/lib64, /usr/share, /var/log).**
- ✓ **2.7 Manage filesystems using the following (check disk usage [df, du], Quotas [edquota, repquota, quotacheck], check and repair filesystems [fsck], Loopback devices [ISO filesystems], NFS [configuration, mount], Swap [mkswap, swapon, swapoff]).**



Most computers' actions are tied very closely to their disk partitions and the files they contain. Web servers must be able to deliver Web files stored on disk, workstations must be able to run applications and store data on disk, and so on. Therefore, it's important that you can manage these files and the filesystems that contain them when you work with a Linux computer. Much of this chapter is devoted to this topic, starting with a look at the underlying hardware and Linux's interfaces to it, then moving on to partition management, and finally looking at the Linux filesystem layout.



Objective 2.7 is covered partly in this chapter and partly in Chapter 10, "Configuring Network Servers."



The term "filesystem" has two meanings. First, it may refer to an organized collection of files, stored in some specific set of directories. For instance, certain filesystem standards for Linux specify in what directories certain types of files reside. Second, "filesystem" may refer to the low-level data structures used to organize files on a hard disk partition or removable disk. Several different filesystems of this second variety exist, such as ext3fs, ReiserFS, and FAT. This chapter covers both types of filesystems; which meaning is intended is usually clear from the context. When it isn't, I clarify by using terms such as "directory structure" or "directory tree" for the first type or "low-level filesystem" for the second variety.

Storage Hardware Identification

Before delving into the details of how Linux manages partitions and files, you should understand some of the basics of storage devices. Several types of storage devices exist, and specific types have characteristics that can influence how Linux interacts with them. Understanding how Linux interacts with the hardware (device filenames, for instance) is also critically important.



Chapter 1, "Getting Started with Linux," describes low-level hardware concerns in more detail.

Types of Storage Devices

Today, many storage devices are in common use on small computers:

Hard disks The most important form of storage for most desktop and server computers is the hard disk. This device consists of one or more spinning platters with magnetic coatings and a read/write head that accesses the data stored on the disk. Hard disks use *Advanced Technology Attachment (ATA)* and *Small Computer System Interface (SCSI)* interfaces, which are explained in more detail shortly, in “Linux Storage Hardware Configuration.” Hard disks have high capacity and are reasonably fast and inexpensive, which means they’re the ideal storage tool for the OS itself and for most user data files.

Removable magnetic disks Removable magnetic disks are much like hard disks, but they can be easily removed from a computer. Examples include floppy disks and Zip disks. These technologies were once extremely common, but they’ve been largely supplanted by removable solid-state and optical devices, which are described shortly. A variant of this approach is to mount normal ATA or SCSI disks in special drive bays to enable easy disk swaps. This configuration can be useful for quick disk backups and recovery.

Optical media Optical media include Compact Disc Read-Only Memory (CD-ROM), CD Recordable (CD-R), CD Re-Writable (CD-RW), DVD, and various recordable DVD variants. These media all use entirely optical methods for storing and reading data. In practice, although they can be read much like removable magnetic disks, optical media require special tools to be written. In fact, not all optical media are recordable—CD-ROMs and DVDs are not recordable. Some media, including CD-Rs and some recordable DVD formats, can be written just once; data cannot be changed once written.

Removable solid-state storage In recent years, solid-state storage devices have grown in popularity. Examples include Compact Flash (CF) cards, Secure Digital (SD) cards, and USB flash drives. These devices can interface with computers much like hard disks or removable magnetic disks and can be treated much like these devices from a software perspective. They differ in their underlying technologies, though; rather than store data on magnetized spinning disks, solid-state storage devices use nonvolatile electronic storage, similar in some ways to ordinary RAM. These technologies are expensive and low in capacity compared to most other removable media, but they have the advantage of compact size and durability, which make them ideal for use in various portable digital devices, such as digital cameras and music players. USB flash cards are extremely popular as transportable storage for desktop and laptop computers; they’ve largely replaced floppy disks in this role.

NVRAM *Nonvolatile RAM (NVRAM)* is a way of storing small amounts of data in chips. Although NVRAM is conceptually similar to removable solid-state storage, it’s used in computers to store small amounts of fixed data, such as BIOS options. NVRAM is generally not removable from the computer and is quite limited in size. Linux provides the means to read and write NVRAM. Typically, this requires use of specialized data access tools.

Magnetic tape Magnetic tape has long been an important storage medium for computers, and it remains an important medium for backups. It has speed, cost, and access limitations

that make it impractical for the day-to-day storage of ordinary files, however. Special software (`mt` in Linux) is required to manipulate magnetic tapes.

Each of these classes of storage device has its own unique place on a Linux system. For a typical laptop, desktop, or server, the hard disk plays the most important role in day-to-day operations, although one or more solid-state and optical devices may be important as well. Aside from tapes and NVRAM, all of these devices are most commonly accessed with the help of a low-level filesystem, although the filesystems that are most useful vary from one device to another. (The upcoming section “Linux Filesystem Options” describes these options in more detail.)

Linux Storage Hardware Configuration

To use a storage device, programs must be able to access it. In most cases, this is done through a low-level filesystem, which is then mounted to a directory—that is, the directory serves as a way to access the files and directories on the filesystem.



The upcoming section “Mounting and Unmounting Partitions” describes how to mount filesystems.

In other cases, you must know the device filename that corresponds to the device. By reading from or writing to this device file, you may access data stored on the hardware. In fact, device filenames are important even for mounted filesystems, because you tell Linux what partition to mount by using a device filename.

Hard disk devices are identified as either ATA (with `/dev/hdx` device filenames, where `x` is a letter from `a` onward) or SCSI (with `/dev/sdx` disk identifiers). This characterization, however, is based on the state of hardware at the time this system was devised. Since that time, both the hardware and the driver situation has become more complex.

Today, the ATA marketplace is divided between *parallel ATA (PATA)* and *serial ATA (SATA)* drives. PATA drives are the original ATA technology, whereas SATA is a newer standard. Most new computers today use SATA drives, but PATA drives are still available. PATA devices usually (but not always) have `/dev/hdx` device filenames, whereas SATA drives usually (but not always) have `/dev/sdx` device filenames. Which is used depends on the driver that’s built into the kernel or loaded first. The general trend is to favor the SCSI-style device filenames. True SCSI devices use the `/dev/sdx` identifiers, as do newer Serial Attached SCSI (SAS) disks. Disks and disk-like devices that use neither ATA nor SCSI, such as USB flash disks, generally acquire SCSI-style identifiers.

In the case of true PATA devices, the master disk on the first disk controller is `/dev/hda`, the slave disk on that controller is `/dev/hdb`, the master disk on the second controller is `/dev/hdc`, and so on. Device letters can be skipped, depending on how disks are configured. For instance, if you have two hard disks, both masters on their chains, they might be known as `/dev/hda` and `/dev/hdc`, with `/dev/hdb` unused.

SCSI disk devices are assigned letters sequentially, beginning with `a`; thus, a system with two SCSI disks will identify them as `/dev/sda` and `/dev/sdb`, even if they have noncontiguous

SCSI ID numbers. The same is true of devices that “pretend” to be SCSI disks, such as SATA or USB flash drives.

Both ATA and SCSI hard disks are commonly broken into partitions. These are identified by numbers after the hardware identifier. For instance, `/dev/hda3` identifies a specific partition on `/dev/hda`, and `/dev/sdb5` identifies a partition on `/dev/sdb`. Partitions are numbered starting with 1. Additional rules and limitations depend on the partitioning system, as described in the upcoming section “Understanding Partitioning Systems.”

The actual order of partitions on a disk need not correspond to their partition numbers. For instance, `/dev/hda3` might appear before `/dev/hda2`. Likewise, gaps may appear in the partition sequence—a disk might have `/dev/hda1` and `/dev/hda3` but not `/dev/hda2` or `/dev/hda4`.

Most removable magnetic media and solid-state storage devices use hard disk device files. These devices look almost exactly like hard disks from Linux’s point of view. One exception is floppy disks. Typically, the first floppy disk is `/dev/fd0`, and the second (if it’s present) is `/dev/fd1`. Most distributions also provide assorted specialized files, such as `/dev/fd0u1440`, which enable you to force the OS to access the device at a given capacity. This can be particularly useful when you’re performing a low-level disk format with the `fdformat` utility, which prepares a floppy disk for use. (You must also create a filesystem on the disk, using `mkfs` or a similar utility, as described later in “Creating New Filesystems.”)

Hard disks are almost always partitioned before use. This is also true of some types of removable magnetic media, such as Zip disks, which use conventional ATA or SCSI disk device files to access the hardware. Other removable magnetic media, such as floppy disks, are conventionally not partitioned. This is just a convention, at least for devices that use hard disk device files. (No device filenames for accessing partitioned floppy disks exist.)

Optical media are unusual because their access devices vary depending on the device interface. ATA devices are handled just like hard disks. For instance, an ATA CD-RW drive that’s configured as the slave device on the first disk controller will be accessed as `/dev/hdb`, just as if it were a hard disk. SCSI optical drives, though, are identified with filenames of the form `/dev/scdx`, where *x* is a number from 0 up. Thus, `/dev/scd0` is typically your SCSI optical device. Linux kernels provide SCSI emulation support, which enables you to access an ATA optical drive as if it were a SCSI model. Thus, an ATA optical disc *might* look like a SCSI one, depending on your kernel configuration. Most distributions set up a symbolic link from `/dev/cdrom` to your primary optical media device, so you may be able to use this filename when specifying your disc. Optical discs are not conventionally partitioned.

Magnetic tape devices are identified using a pair of device files, whose names differ from ATA to SCSI. For an ATA tape device, `/dev/htx` and `/dev/nhtx`, where *x* is a number from 0 up, identify the tape device. The `/dev/htx` file, when accessed, causes the tape to automatically rewind after every operation; the `/dev/nhtx` file, by contrast, is nonrewinding, which can be handy if you want to store multiple backups on a single tape. The SCSI device filenames take a similar form: `/dev/stx` and `/dev/nstx`. Because few systems have multiple tape devices, chances are you’ll see the 0-numbered ones only, such as `/dev/st0` and `/dev/nst0`. Typically, tape devices are accessed like ordinary files—you pass their filenames to backup programs as if they were disk files.

NVRAM is accessed through the `/dev/nvram` file. This file contains very precisely structured data, so you should *not* try to access it directly. Instead, you can use a utility, such as NVRAM Wakeup (<http://sourceforge.net/projects/nvram-wakeup>) or `tpb` (<http://www.nongnu.org/tpb>). These programs provide specialized functionality to read or write some or all of the NVRAM data.



Never attempt to write a filesystem to `/dev/nvram` or otherwise write to it without the help of a program designed to do so. Even using such a program designed for a CPU or BIOS other than the one you're using could render your system *unbootable*!

Planning Disk Partitioning

Hard disks can be broken into logical chunks known as *partitions*. In Windows, partitions correspond to drive letters (C:, D:, and so on). In Linux, partitions are mounted at particular points in the Linux directory tree, so they're accessible as subdirectories. Before installing Linux, it's a good idea to give some thought to how you'll partition your hard disk. A poor initial partitioning scheme can become awkward because you'll run out of space in one partition when another has lots of available space or because the partition layout ties your hands in terms of achieving particular goals.

Understanding Partitioning Systems

Partitions are defined by data structures that are written to specified parts of the hard disk. Several competing systems for defining these partitions exist. On x86 and x86-64 hardware, the most common method up until 2009 has been the *master boot record (MBR)* partitioning system, so called because it stores its data in the first sector of the disk, which is also known as the MBR. The MBR system, however, is limited to partitions and partition placement of 2 terabytes (TB), at least when using the nearly universal sector size of 512 bytes. The successor to MBR is the *GUID partition table (GPT)* partitioning system, which has much higher limits and certain other advantages. The tools and methods for manipulating MBR and GPT disks differ from each other, although there's substantial overlap.



Still more partitioning systems exist, and you may run into them from time to time. For instance, Macintoshes that use PowerPC CPUs generally employ the Apple Partition Map (APM), and many Unix variants employ Berkeley Standard Distribution (BSD) disk labels. You're most likely to encounter MBR and GPT disks, so those are the partitioning systems covered in this book. Details for other systems differ, but the basic principles are the same.

MBR Partitions

The original x86 partitioning scheme allowed for only four partitions. As hard disks increased in size and the need for more partitions became apparent, the original scheme was extended in a way that retained backward compatibility. The new scheme uses three partition types:

- *Primary partitions*, which are the same as the original partition types
- *Extended partitions*, which are a special type of primary partition that serves as a placeholder for the next type
- *Logical partitions*, which reside within an extended partition

For any one disk, you're limited to four primary partitions, or three primary partitions and one extended partition. Many OSs, such as DOS, Windows, and FreeBSD, *must* boot from primary partitions, and because of this, most hard disks include at least one primary partition. Linux, however, is not so limited, so you could boot Linux from a disk that contains no primary partitions, although in practice few people do this.

The primary partitions have numbers in the range of 1–4, whereas logical partitions are numbered 5 and up. Gaps can appear in the numbering of MBR primary partitions; however, such gaps cannot exist in the numbering of logical partitions. That is, you can have a disk with partitions numbered 1, 3, 5, 6, and 7 but not 1, 3, 5, and 7—if partition 7 exists, there must be a 5 and a 6.

In addition to holding the partition table, the MBR data structure also holds the primary boot loader—the first disk-loaded code that the CPU executes when the computer boots. Thus, the MBR is extremely important and sensitive. Because the MBR exists only in the first sector of the disk, it's vulnerable to damage; accidental erasure will make your disk unusable unless you have a backup.



You can back up your MBR by typing **dd if=/dev/sda of=~sda-bu.img bs=512 count=1** (or similar commands to specify another disk device or backup file). You can then copy the backup file (~sda-bu.img in this example) to a removable disk or another computer for safekeeping. Be sure to type this command correctly, though; reversing the **if=** and **of=** options will destroy your MBR! Note that backing up your MBR in this way won't back up your logical partitions, just your primaries and the extended partition, if you have one.

Although the MBR data structure has survived for a quarter century, its days are numbered because it's not easily extensible beyond 2TB disks. Thus, a new system is needed.

GPT Partitions

GPT is part of Intel's Extensible Firmware Interface (EFI) specification, which is intended as a replacement for the BIOS that most x86 and x86-64 computers use. GPT can be used on computers that don't use EFI, though, and GPT is the preferred partitioning system for disks bigger than 2TB.

GPT employs three data structures, two of which have backups:

Protective MBR An MBR exists on a GPT-partitioned disk, with the purpose of deterring creation of ordinary MBR partitions. The protective MBR, as it's called, defines a single partition with a type code of 0xEE (EFI GPT). This partition spans the size of the disk or 2TB, whichever is smaller, thus signaling that the disk is a GPT disk and (it is hoped) minimizing accidental damage should a GPT-unaware utility be used to access the disk. The protective MBR may also hold a boot loader in its code area.

Header The header defines various GPT metadata, such as the size of the partition table, the locations of the partition tables, and cyclic redundancy check (CRC) checksums to help system software detect data corruption.

Partition table The partition table defines actual partitions. On most disks, the partition table can define up to 128 partitions. GPT does away with the primary/extended/logical distinction of MBR, which simplifies certain disk administration tasks. In Linux, GPT partitions are numbered starting with 1, and they need not be consecutive, so you could have partitions numbered 2, 5, 23, and 120. (Many disk utilities will automatically change such a sequence to 1, 2, 3, and 4, however.)

The GPT header and partition table are duplicated: one copy appears at the start of the disk, and the other copy appears at the end of the disk. This redundancy, along with the CRCs in the header, simplifies some types of data recovery.

GPT's main drawback is that support for it is relatively immature. The `fdisk` utility (described shortly in “Partitioning Tools”) doesn't work with GPT disks. The old LILO boot loader and some versions of GRUB also don't support it. GPT support must also be enabled in your kernel. (Most distributions do so by default.) The situation is worse in some OSs—particularly older ones. Nonetheless, you should be at least somewhat familiar with GPT because of MBR's inability to handle disks larger than 2TB.

Linux Partition Requirements

To Linux, there's very little difference between the primary and logical partitions or even between MBR and GPT partitions. As noted earlier, there are numbering implications for the different partitioning schemes, and you should be familiar with them.

Some administrators use a primary Linux boot partition because a conventional x86 boot loader can boot only from a primary partition. When the computer does so, it runs code in the boot sector of the boot partition. Typically, Linux places a special boot loader program in this location. The *Grand Unified Boot Loader (GRUB)* and the *Linux Loader (LILO)* are the two boot loaders most commonly found on x86 Linux systems. Alternatively, GRUB or LILO can reside directly in the MBR, which is more direct but leaves the boot loader more vulnerable to being wiped out should some other utility rewrite the MBR. Placing GRUB in the protective MBR is the only option if you're booting a BIOS-based system from a GPT disk.



Non-x86 distributions need boot loaders, too, but they're different from x86 boot loaders in various details. Sometimes a boot loader such as GRUB or LILO is ported or copied on non-x86 distributions. The IA-64 platform uses a boot loader called ELILO, for instance. Other times, a completely new boot loader is used, such as Yaboot for PowerPC systems.

At a bare minimum, Linux needs a single partition to install and boot. This partition is referred to as the *root partition*, or */*. This partition is so called because it holds the root directory, which lies at the “root” of the directory “tree”—all files on the computer are identified relative to the root directory. The root partition also stores directories, such as */etc* and */bin*, that fall off the root directory and in which other files reside. Some of these directories can serve as *mount points*—directories to which Linux attaches other partitions. For instance, you might mount a partition on */home*.



One important directory in Linux is */root*, which serves as the system administrator's home directory—the system administrator's default program settings and so on go here. The */root* directory is not to be confused with the root (*/*) directory.

One partitioning strategy that's common on high-performance systems is a Redundant Array of Independent Disks (RAID). In a RAID configuration, partitions on separate physical hard disks are combined together to provide faster performance, greater reliability, or both. Some Linux distributions provide RAID options in their initial installation procedures, but others don't. RAID configuration is fairly advanced and is covered later in “Using RAID.” If you're new to Linux, it's best to avoid RAID configurations on your first installation. You might try implementing a RAID configuration on subsequent installations.

Common Optional Partitions

In addition to the root partition, many system administrators like creating other partitions. These are some advantages that come from splitting an installation into multiple partitions rather than leaving it as one monolithic root partition:

Multiple disks When you have two or more hard disks, you *must* create separate partitions—at least one for each disk. For instance, one disk might host the root directory, and the second might hold */home*. Also, removable disks (floppies, CD-ROMs, and so on) must be mounted as if they were separate partitions.

Better security options By breaking important directories into separate partitions, you can apply different security options to different partitions. For instance, you might make */usr* read-only, which reduces the chance of accidental or intentional corruption of important binary files.

Data overrun protection Some errors or attacks can cause files to grow to huge sizes, which can potentially crash the system or cause serious problems. Splitting key directories into separate partitions guarantees that a runaway process in such a directory won't cause problems for processes that rely on the ability to create files in other directories. This makes it easier to recover from such difficulties. On the downside, splitting partitions up makes it more likely that a file will legitimately grow to a size that fills the partition.

Disk error protection Disk partitions sometimes develop data errors, which are data structures that are corrupted or a disk that has developed a physically bad sector. If your system consists of multiple partitions, such problems will likely be isolated to one relatively small partition, which can make data recovery easier or more complete.

Backup If your backup medium is substantially smaller than your hard disk, breaking up your disk into chunks that fit on a single medium can simplify your backup procedures.

Ideal filesystems Sometimes, one filesystem works well for some purposes but not for others. You might therefore want to break the directory tree into separate partitions so that you can use multiple filesystems.

So, what directories are commonly split off into separate partitions? Table 6.1 summarizes some popular choices. Note that typical sizes for many of these partitions vary greatly depending on how the computer is used. Therefore, it's impossible to make recommendations on partition size that will be universally acceptable.

TABLE 6.1 Common Partitions and Their Uses

Partition (Mount Point)	Typical Size	Use
Swap (not mounted)	1.5–2 times system RAM size	Serves as an adjunct to system RAM; is slow but enables the system to run more or larger programs.
/home	200MB–1TB	Holds users' data files. Isolating it on a separate partition preserves user data during a system upgrade. Size depends on number of users and their data storage needs.
/boot	50–200MB	Holds critical boot files. Creating as a separate partition allows for circumventing limitations of older BIOSs and boot loaders on hard disks over 8GB.
/usr	500MB–15GB	Holds most Linux program and data files.
/usr/local	100MB–5GB	Holds Linux program and data files that are unique to this installation, particularly those that you compile yourself.

TABLE 6.1 Common Partitions and Their Uses (*continued*)

Partition (Mount Point)	Typical Size	Use
/opt	100MB–5GB	Holds Linux program and data files that are associated with third-party packages, especially commercial ones.
/var	100MB–500GB	Holds miscellaneous files associated with the day-to-day functioning of a computer. These files are often transient in nature. This directory is most often split off as a separate partition when the system functions as a server that uses the /var directory for server-related files like mail queues.
/tmp	100MB–50GB	Holds temporary files created by ordinary users.
/mnt	N/A	/mnt isn't itself a separate partition; rather, it or its subdirectories are used as mount points for removable media like USB flash drives or CD-ROMs.
/media	N/A	Holds subdirectories that may be used as mount points for removable media, much like /mnt or its subdirectories.

Some directories—/etc, /bin, /sbin, /lib, and /dev—should *never* be placed on separate partitions. These directories host critical system configuration files or files without which a Linux system cannot function. For instance, /etc contains /etc/fstab, the file that specifies what partitions correspond to what directories, and /bin contains the mount utility that's used to mount partitions on directories.



The 2.4.x and newer kernels include support for a dedicated /dev filesystem, which obviates the need for files in an actual /dev directory, so in some sense, /dev can reside on a separate filesystem, although not a separate partition.

In addition to the separate partitions specified in Table 6.1, you should be aware of the existence and purpose of several other directories and subdirectories on a Linux system. Most importantly, The /etc directory holds most system configuration files, /bin holds binary files that may be run by any user, /sbin holds binary files that are typically run only by root, and /lib holds libraries that may be used by programs stored in /bin or /sbin. (Some distributions include directories called /lib32 and /lib64, which hold 32-bit and 64-bit libraries, respectively. This helps x86-64 distributions to support both 32-bit and 64-bit programs.) All

of these directories, with the exception of `/etc`, have namesakes within `/usr`, as in `/usr/bin` and `/usr/lib`. The binary and library directories that exist directly off of the root directory hold critical system files—those that Linux needs to boot and operate as a bare-bones system. The namesakes of these directories under `/usr` hold program files and libraries that are less critical for basic functioning, such as word processors, Web browsers, and the X Window System. The `/usr/share` directory also deserves mention; it holds miscellaneous shared data files, such as fonts, as well as ancillary data files for specific programs. The `/proc` directory, described in more detail in Chapter 3, holds pseudo-files that provide information on, and in some cases a means to control, low-level hardware devices.

Linux Filesystem Options

Linux's standard filesystem for most of the 1990s was the *second extended filesystem* (*ext2* or *ext2fs*), which was the default filesystem for most distributions. Ext2fs supports all the features required by Linux (or by Unix-style OSs in general) and is well tested and robust.



Real World Scenario

When to Create Multiple Partitions

One problem with splitting off lots of separate partitions, particularly for new administrators, is that it can be difficult to settle on appropriate partition sizes. As noted in Table 6.1, the appropriate size of various partitions can vary substantially from one system to another. For instance, a workstation is likely to need a fairly small `/var` partition (say, 100MB), but a mail or news server might need a `/var` partition that's gigabytes in size. Guessing wrong is annoying. You'll need to resize your partitions (which is tedious and dangerous) or set up symbolic links between partitions so that subdirectories on one partition can be stored on other partitions.

For this reason, I generally recommend that new Linux administrators try simple partition layouts first. The root (`/`) partition is required, and swap is a very good idea. Beyond this, `/boot` can be very helpful on hard disks of more than 8GB with older distributions or BIOSs but is seldom needed with computers or distributions made since 2000. An appropriate size for `/home` is often relatively easy for new administrators to estimate, so splitting it off generally makes sense. Beyond this, I recommend that new administrators proceed with caution.

As you gain more experience with Linux, you may want to break off other directories into their own partitions on subsequent installations or when upgrading disk hardware. You can use the `du` command to learn how much space is used by files within any given directory.

Ext2fs has one major problem, though: if the computer is shut down improperly (because of a power outage, system crash, or the like), it can take several minutes for Linux to verify an ext2fs partition's integrity when the computer reboots. This delay is an annoyance at best,

and it is a serious problem on mission-critical systems such as major servers. The solution is implemented in what's known as a *journaling filesystem*. Such a filesystem keeps a record of changes it's about to make in a special journal log file. Therefore, after an unexpected crash, the system can examine the log file to determine what areas of the disk might need to be checked. This design makes for very fast checks after a crash or power failure—a few seconds at most, typically. The following are the five most common journaling filesystems for Linux:

- The *third extended filesystem (ext3fs)*, which is derived from ext2fs and is the most popular journaling filesystem for Linux in 2009
- The *fourth extended filesystem (ext4fs)*, which is the next generation in this filesystem line, adds support for larger disks and other advanced features
- *ReiserFS*, which was added as a standard component to the 2.4.1 kernel
- The *Extent Filesystem*, or XFS, which was originally designed for Silicon Graphics' IRIX OS
- The *Journaled Filesystem*, or JFS, which IBM developed for its AIX and OS/2

Of these five, XFS and JFS are the most advanced, but ext3fs and ReiserFS are the most stable and popular. Ext4fs promises to be the best of both worlds, but it's still fairly new—it was added as a stable filesystem only with the 2.6.29 kernel. A derivative of the current ReiserFS, Reiser4, is under development.



The Linux swap partition doesn't use a filesystem per se. Linux does need to write some basic data structures to this partition in order to use it as swap space, but this isn't technically a filesystem because no files are stored within it.

Linux also supports many non-Linux filesystems, including the following:

- The File Allocation Table (FAT) filesystem used by DOS and Windows and its long-filename extension, Virtual FAT (VFAT)
- The New Technology Filesystem (NTFS) used by Windows NT/200x/XP/Vista
- The High-Performance Filesystem (HPFS) used by OS/2
- The Unix Filesystem (UFS; also known as the Fast Filesystem, or FFS) used by various versions of Unix
- The Hierarchical Filesystem (HFS) used by Mac OS
- ISO-9660 and Joliet filesystems used on CD-ROMs
- The Universal Disk Format (UDF), which is the up-and-coming successor to ISO-9660 for optical discs

Most of these filesystems are useful mainly in dual-boot configurations—for instance, to share files between Linux and Windows. Some—particularly FAT, ISO-9660, Joliet, and UDF—are useful for exchanging files between computers on removable media. As a general rule, these filesystems can't hold critical Linux files because they lack necessary filesystem features. There are exceptions, though: Linux sports extensions for cramming necessary

information into FAT and HPFS partitions, UFS was designed for storing Unix filesystem features in the first place, and the Rock Ridge extensions add the necessary support to ISO-9660.

It's usually best to use a journaling filesystem for Linux partitions. As a general rule, any of the current crop of journaling filesystems works well, at least with recent (late 2.4.x or newer) kernels. The best tested under Linux are ext3fs and ReiserFS. ReiserFS versions of 3.5 and older have a 2GB file-size limit, but this limit is raised to 16TB for ReiserFS 3.6 and newer. XFS and ext3fs have the widest array of filesystem support tools, such as versions of `dump` and `restore` for creating and restoring backups. All of the journaling filesystems support a flexible security system known as *access control lists (ACLs)*, which are particularly important if your computer functions as a Samba server to Windows NT/200x/XP/Vista clients. All modern Linux distributions support ext2fs and ext3fs out of the box, and many support at least one or two others as well. You can use filesystems that aren't supported by your distribution on any Linux system, but you'll need to jump through some hoops by installing software and perhaps recompiling your kernel. You should use non-Linux filesystems mainly for data exchange with non-Linux systems.

Partitioning Tools

In order to create partitions, you use a partitioning tool. Dozens of such tools are available, but only a few are reasonable choices when you're installing a Linux system:

DOS's `FDISK` Microsoft's DOS and Windows ship with a simple partitioning tool known as `FDISK` (for "fixed disk"). This program is inflexible and uses a crude text-based user interface, but it's readily available and can create partitions that Linux can use. (You'll probably have to modify the partition type codes using Linux tools in order to use DOS-created partitions, though.)

Linux's `fdisk` Linux includes a partitioning tool that's named after the DOS program, but the Linux tool's name is entirely lowercase, whereas the DOS tool's name is usually written in uppercase. Linux's `fdisk` is much more flexible than DOS's `FDISK`, but it also uses a text-based user interface. If you have an existing Linux emergency disk, you can use it to create partitions for Linux before installing the OS.

GPT `fdisk` Linux's `fdisk` is designed to manage MBR partitions. A similar tool for handling GPT partitions is `GPT fdisk` (aka `gdisk`; <http://www.rodsbooks.com/gdisk>). Note that the author of this Study Guide is the creator of `GPT fdisk`.

Linux install-time tools Most Linux installation utilities include partitioning tools. Sometimes the installers simply call `fdisk`, but other times they provide GUI tools that are much easier to use. If you're installing a Linux-only system, using the installer's tools is probably the best course of action.

GNU Parted GNU Parted (<http://www.gnu.org/software/parted>) is an open source tool that can manage both MBR and GPT disks. It can create, resize, and move various partition types, such as FAT, ext2fs, ext3fs, ReiserFS, and Linux swap. GNU Parted runs from Linux and provides a text-only user interface.

QTParted This program, headquartered at <http://qtparted.sourceforge.net>, provides a GUI front-end to `libparted`, which is the “brains” behind GNU Parted. As such, it has most of GNU Parted’s capabilities, including its capacity to handle both MBR and GPT disks.

GNOME Partition Editor This program, which is also known as GParted, is another GUI front-end to `libparted`. It’s based at <http://gparted.sourceforge.net>.

In theory, partitions created by any tool may be used in any OS, provided the tools and OSs all use the same partition table type (MBR or GPT). In practice, though, OSs sometimes object to unusual features of partitions created by certain partitioning tools. Therefore, it’s usually best to take one of two approaches to disk partitioning:

- Use a cross-platform partitioning tool. Tools based on `libparted` are good in this regard. Such tools tend to create partitions that are inoffensive to all major OSs.
- Use each OS’s partitioning tool to create that OS’s partitions.

Partition Management and Maintenance

As just described, Linux systems store their data on disk partitions. Most partitions hold low-level filesystems. Creating partitions and preparing them to hold data are critically important tasks in using disks. To create partitions on a new disk, you use a disk partitioning tool, such as `fdisk` or GNU Parted. You then create a new low-level filesystem on the partition using `mkfs`. (GNU Parted can do this, too.) Sometimes filesystems develop errors, in which case checking their integrity with `fsck` is critically important.

Some partitions hold *swap space* rather than filesystems. Swap space can function as a stand-in for RAM when the system runs out of physical memory. Having adequate swap space is important for overall system functioning, and being able to manage it will help your system work well.

Creating Partitions

Several tools for manipulating partitions exist, as described earlier in “Partitioning Tools.” The Linux+ objectives emphasize `fdisk` and GNU Parted, and these are the most popular text-based tools, so they’re the ones described here. If you prefer GUI tools, QTParted and the GNOME Partition Editor implement GNU Parted’s functionality but with a GUI face.



If partitions are in use on a disk, it’s sometimes necessary to reboot the computer after making changes to the disk’s partition table. Failure to do so will cause Linux to continue to use the old partitions, potentially resulting in data loss. Partitioning tools typically warn you when they detect a need to reboot, so watch for that message.

Using *fdisk* to Create Partitions

The *fdisk* program is a fairly basic text-mode tool for manipulating partitions. If you use *fdisk*, you'll need to manually create filesystems, configure RAID or LVM, or set up swap space afterward.



Linux on non-MBR systems may not use a tool called *fdisk*. For instance, PowerPC versions of Linux, which employ the APM partitioning system, use a tool called *pdisk*. Further complicating matters, some PowerPC Linux distributions call their *pdisk* programs *fdisk*. If your computer uses GPT, the *gdisk* program works much like *fdisk*, although a few details differ. If you're using a non-MBR system, consult the documentation for your distribution and its disk-partitioning tool. If you prefer, you can use GNU Parted on a wide variety of partition table types, bypassing *fdisk* or its workalikes entirely.

Linux's *fdisk* is a text-based tool that requires you to type one-letter commands. You can obtain a list of commands by typing **?** or **m** at the *fdisk* prompt. Table 6.2 describes the most important *fdisk* commands.

TABLE 6.2 *fdisk* Commands

Command	Description
d	Deletes a partition
l	Lists partition type codes
m	Displays available commands
n	Creates a new partition
o	Creates a new partition table, wiping out all current partitions
p	Displays (prints) the partition layout
q	Quits without saving changes
t	Changes a partition's type code
w	Writes (saves) changes and quits

To start *fdisk*, type its name followed by the Linux device filename associated with your disk device, as in ***fdisk /dev/hdb***. When you first start *fdisk*, the program displays its prompt. It's often helpful to type **p** at this prompt to see the current partition layout,

as shown in Figure 6.1. This will help you verify that you're operating on the correct disk, if you have more than one hard disk. It will also show you the device IDs of the existing disk partitions.

The `fdisk` program identifies the extended partition, if it's present, in the System column of its output, as shown in Figure 6.1; these partitions may be labeled as Extended or Win95 Ext 'd' (LBA); Linux treats both types identically. Primary and logical partitions are not explicitly identified as such; you must use the partition number, as described earlier in "Linux Storage Hardware Configuration," to identify the partition type.

You can use the commands outlined in Table 6.2 to alter a disk's partition layout, but be aware that your changes are potentially destructive. Deleting partitions will make their data inaccessible. Some commands require you to enter additional information, such as partition numbers or sizes for new partitions. For instance, the following sequence illustrates the commands associated with adding a new logical partition that's 2GB in size:

```
Command (m for help): n
Command action
  l   logical (5 or over)
  p   primary partition (1-4)
1
First cylinder (519-784, default 519): 519
Last cylinder or +size or +sizeM or +sizeK (519-784, default 784): +2G
```

FIGURE 6.1 As a text-based program, `fdisk` can be run in text mode or in a terminal window, as shown here.

```

root@speaker: ~ - Shell - Konsole
Session Edit View Bookmarks Settings Help

and could in certain setups cause problems with:
1) software that runs at boot time (e.g., old versions of LILO)
2) booting and partitioning software from other OSs
   (e.g., DOS FDISK, OS/2 FDISK)

Command (m for help): p

Disk /dev/sda: 160.0 GB, 160041885696 bytes
255 heads, 63 sectors/track, 19457 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Disk identifier: 0x8e0cb6b5

   Device Boot      Start         End      Blocks   Id  System
/dev/sda1           1         1247       10016496   17  Hidden HPFS/NTFS
/dev/sda2          1248         6455       41833260   17  Hidden HPFS/NTFS
/dev/sda3           6456       15556       73103782+   5  Extended
/dev/sda4            *       15557       19457       31334782+  af  Unknown
/dev/sda5           6456         8203       14040778+   17  Hidden HPFS/NTFS
/dev/sda6           8204       11453       26105593+   b  W95 FAT32
/dev/sda7          11454       11645       1542208+   82  Linux swap / Solaris
/dev/sda8          11646       11665       160618+    83  Linux
/dev/sda9          11666       15556       31254426   8e  Linux LVM

Command (m for help):
```

You can enter the partition size in terms of cylinder numbers or as a size in bytes, kilobytes, megabytes, or gigabytes (which isn't mentioned in the prompt but does work). When you've made your changes, type **w** to write them to disk and exit. If you make a mistake, type **q** immediately; doing this will exit from `fdisk` without committing changes to disk.

Using GNU Parted to Create Partitions

GNU Parted is a more sophisticated partitioning tool than `fdisk`. GNU Parted supports several partition table types, including both MBR and GPT; it enables creation of filesystems during partition creation; and it can resize and move both partitions and the filesystems they contain. GNU Parted doesn't provide full support for all filesystems, though, and the details of what filesystems it does support varies from one build to another. Ext2fs, ext3fs, and FAT are almost always supported, but beyond that you may need to experiment.

To begin using GNU Parted, type the program name followed by the device filename associated with the disk you want to modify, as in `parted /dev/hdb`. You'll be greeted by a (parted) prompt, at which you can type GNU Parted commands, the most important of which are summarized in Table 6.3. Note that many commands have optional parameters. In most cases, if you omit these, GNU Parted will prompt you for the relevant information.

TABLE 6.3 GNU Parted Commands

Command	Description
<code>help [COMMAND]</code>	Displays information on all commands or on the one specified.
<code>mklabel [type]</code>	Creates a new partition table. The <i>type</i> is a type code, such as <code>msdos</code> for MBR or <code>gpt</code> for GPT.
<code>mkfs [number [fstype]]</code>	Creates a new filesystem on an existing partition. You can specify a partition <i>number</i> and a filesystem type (<i>fstype</i>).
<code>mkpart [type [fstype]] [start [end]]</code>	Creates a new partition. The <i>type</i> is primary, extended, or logical; the <i>fstype</i> is the filesystem type the partition will hold; and the <i>start</i> and <i>end</i> values are specified in bytes, kilobytes, megabytes, or gigabytes. (You use letter suffixes, as in 2G, to specify units.)
<code>mkpartfs [type [fstype]] [start [end]]</code>	Creates a new partition with a filesystem. This command works just like <code>mkpart</code> , but it creates a filesystem in the partition.
<code>move [number [start [end]]]</code>	Moves a partition.
<code>name [number [name]]</code>	Names a partition. This option is not meaningful for MBR disks, although it does set the partition name for GPT disks.

TABLE 6.3 GNU Parted Commands (*continued*)

Command	Description
<code>print [free <i>number</i> all]</code>	Displays information. If given by itself, the partition table is shown. The <i>free</i> suboption displays a summary of free (unpartitioned) space on the disk. If you specify a <i>number</i> , basic information on the filesystem in the specified partition is displayed. If you use the <i>all</i> suboption, information on all disk devices is shown.
<code>quit</code>	Exits from the program.
<code>resize [<i>number</i> [<i>start</i> [<i>end</i>]]]</code>	Resizes the specified partition and its filesystem to the new size (defined by the <i>start</i> and <i>end</i> points).
<code>rm [<i>number</i>]</code>	Deletes the specified partition.
<code>select [<i>device</i>]</code>	Switches from the current device to the specified one.



GNU Parted implements its changes immediately. With `fdisk`, if you make a mistake, you can use the `q` command to exit without altering the disk. With GNU Parted, you have no such way out. You should therefore be extra careful when using GNU Parted.

As an example of GNU Parted in action, consider using it to create a new (second) partition on a 2GB USB flash disk:

```
# parted /dev/sdc
GNU Parted 1.7.1
Using /dev/sdc
Welcome to GNU Parted! Type 'help' to view a list of commands.
(parted) print

Disk /dev/sdc: 2038MB
Sector size (logical/physical): 512B/512B
Partition Table: msdos

Number  Start   End     Size    Type    File system  Flags
 1      0.51kB 1000MB 1000MB  primary fat32        lba

(parted) mkpartfs
Partition type? primary/extended? primary
```

```
File system type? [ext2]?
Start? 1000MB
End? 2038MB
(parted)
```

In this example, the `print` command displayed the current partition table to verify that the correct disk was being used. Upon issuing the `mkpartfs` command without extra options, GNU Parted prompted for values: the partition type, the file system type (giving no response caused the default value of `ext2` to be used), and the start and end points. Unlike `fdisk`, which deals in cylinders by default, GNU Parted expects values in other units—megabytes in this example, although kilobytes or gigabytes could have been used. Although the display was eventually replaced by the final `(parted)` prompt, the program briefly displayed a summary of what it did as it created the partition and then the filesystem within it.

Creating New Filesystems

Just creating partitions isn't enough to make them useful in Linux. To make them useful, you must create a filesystem on the partition (a task that's also sometimes called "formatting" a partition). If you use GNU Parted, you may be able to create a filesystem when you create a partition. If GNU Parted doesn't support your filesystem type, if you prefer to do it manually, or if you use `fdisk` to create partitions, you can use the `mkfs` program to do the job. This tool has the following syntax:

```
mkfs [-V] [-t fstype] [options] device [blocks]
```



`mkfs` is actually just a front-end to tools that do the real work for specific filesystems, such as `mke2fs` (also known as `mkfs.ext2`). You can call these tools directly if you prefer, although their syntax may vary from that of `mkfs`.

The `mkfs` parameters can be used to perform several tasks, as summarized in Table 6.4.

TABLE 6.4 `mkfs` Options

Option	Description
<code>-V</code>	This option causes <code>mkfs</code> to generate verbose output, displaying additional information on the filesystem-creation process.
<code>-t fstype</code>	You specify the filesystem type with the <code>-t fstype</code> option. Common values for <code>fstype</code> include <code>ext2</code> (for <code>ext2fs</code>), <code>ext3</code> (for <code>ext3fs</code>), <code>ext4</code> (for <code>ext4fs</code>), <code>reiserfs</code> (for ReiserFS), <code>xf</code> s (for XFS), <code>jfs</code> (for JFS), and <code>msdos</code> (for FAT). Some other options are available as well.


TABLE 6.4 mkfs Options (*continued*)

Option	Description
filesystem-specific options	Many of the underlying tools upon which mkfs relies support their own options. You can pass these along, but details vary from one filesystem to another. The <code>-c</code> and <code>-v</code> options are common; these perform low-level hardware checks when creating the filesystem and generate verbose output, respectively.

device is the name of the device on which you want to create the filesystem, such as `/dev/sda5` or `/dev/fd0`. You should *not* normally specify an entire hard disk here (such as `/dev/sda` or `/dev/hdb`). One exception might be if it's a removable disk, but even these are often partitioned.


The *blocks* parameter sets the size of the filesystem in blocks (usually 1,024 bytes in size). You don't normally need to specify this value, since `mkfs` can determine the filesystem size from the size of the partition.

Depending on the size and speed of the disk device, the filesystem-creation process is likely to take anywhere from under a second to a minute or two. If you specify a filesystem check (which is often a good idea, particularly on new or very old disks), this process can take several minutes or possibly over an hour. Once it's done, you should be able to mount the filesystem and use it to store files.



The filesystem-creation process is inherently destructive. If you accidentally create a filesystem in error, it will be impossible to recover files from the old filesystem unless you're very knowledgeable about filesystem data structures or you pay somebody with such knowledge. Recovery costs are apt to be very high.

As noted earlier, `mkfs` is just a front-end to other utilities, which are sometimes called directly instead. For instance, you might call the `mkreiserfs` utility to prepare a ReiserFS partition, `mkfs.ext3` to prepare an ext3fs partition, or `mkdosfs` to prepare a FAT partition or floppy disk. Check the `/sbin` directory for files whose names begin with `mkfs` to see what other filesystem-specific `mkfs` tools exist on your system.



The presence of a filesystem-creation tool on your computer doesn't necessarily mean that you'll be able to read and write the filesystem on your computer. Mounting a filesystem requires appropriate kernel support, which can be compiled and installed independently of the filesystem's `mkfs.fstype` tool.

Checking a Filesystem for Errors

Creating partitions and filesystems are tasks you’re likely to perform every once in a while—say, when adding a new hard disk or making major changes to an installation. Another task is much more common: checking a filesystem for errors. Bugs, power failures, and mechanical problems can all cause the data structures on a filesystem to become corrupted. If these problems are left unchecked, they can cause severe data loss. For this reason, Linux includes tools for verifying a filesystem’s integrity and for correcting any problems that might exist. The main tool you’ll use for this purpose is called `fsck`. Like `mkfs`, `fsck` is actually a front-end to other tools, such as `e2fsck` (aka `fsck.ext2` and `fsck.ext3`). The syntax for `fsck` is as follows:

```
fsck [-sACVRTNP] [-t fstype] [--] [fsck-options] filesystems
```

The more common parameters to this command enable you to perform useful actions, as summarized in Table 6.5.

TABLE 6.5 `fsck` Options

Option	Description
-A	This option causes <code>fsck</code> to check all the filesystems marked for routine checks in <code>/etc/fstab</code> . This option is normally used in system startup scripts.
-C	This option displays a text-mode progress indicator of the check process. Most filesystem check programs don’t support this feature, but <code>e2fsck</code> does.
-V	Verbose output can be produced by using this option.
-N	This option tells <code>fsck</code> to display what it would normally do, without actually doing it.
-t <i>fstype</i>	Normally, <code>fsck</code> determines the filesystem type automatically. You can force the type with this flag, though. If used in conjunction with <code>-A</code> , this causes the system to check only the specified filesystem types, even if others are marked to be checked. If <i>fstype</i> is prefixed with <code>no</code> , then all filesystems <i>except</i> the specified type are checked.
Filesystem-specific options	Filesystem check programs for specific filesystems often have their own options. The <code>fsck</code> command passes options it doesn’t understand, or those that follow a double dash (<code>--</code>), to the underlying check program. Common options include <code>-a</code> or <code>-p</code> (perform an automatic check), <code>-r</code> (perform an interactive check), and <code>-f</code> (force a full filesystem check even if the filesystem initially appears to be clean).

Normally, you run `fsck` with only the filesystem name, as in `fsck /dev/sda6`. You can add options as needed, however. Check the `fsck` man page for less common options.



Run `fsck` *only* on filesystems that are not currently mounted or that are mounted in read-only mode. Changes written to disk during normal read/write operations can confuse `fsck` and result in filesystem corruption.

Linux runs `fsck` automatically at startup on partitions that are marked for this in `/etc/fstab`, as described later in the section “Defining Standard Filesystems.” The normal behavior of `e2fsck` causes it to perform just a quick cursory examination of a partition if it’s been unmounted cleanly. The result is that the Linux boot process isn’t delayed because of a filesystem check unless the system wasn’t shut down properly. A couple of exceptions to this rule exist, however: `e2fsck` forces a check if the disk has gone longer than a certain amount of time without checks (normally six months) or if the filesystem has been mounted more than a certain number of times since the last check (normally 20). Therefore, you will occasionally see automatic filesystem checks of `ext2fs`, `ext3fs`, and `ext4fs` partitions even if the system was shut down correctly.

Journaling filesystems do away with filesystem checks at system startup even if the system was not shut down correctly. These filesystems keep a log of pending operations on the disk so that in the event of a power failure or system crash the log can be checked and its operations replayed or undone to keep the filesystem in good shape. This action is automatic when mounting such a filesystem. Nonetheless, these filesystems still require check programs to correct problems introduced by undetected write failures, bugs, hardware problems, and the like. If you encounter odd behavior with a journaling filesystem, you might consider unmounting it and performing a filesystem check—but be sure to read the documentation first. Some Linux distributions do odd things with some journaling filesystem check programs. Most notably, ReiserFS should not normally be checked at system startup, so distributions sometimes create special scripts or link `/sbin/fsck.reiserfs` to `/bin/true` to avoid problems. Such configurations speed system boot times should ReiserFS partitions be marked for automatic checks, but they can be confusing if you need to manually check the filesystem. If this is the case, run `/sbin/reiserfsck` to do the job.

Adding Swap Space

Linux enables you to run programs that consume more memory than you have RAM in your system. It does this through the use of swap space, which is disk space that Linux treats as an extension of RAM. When your RAM fills with programs and their data, Linux moves some of this information to its swap space, freeing actual RAM for other uses. This feature, which is common on modern operating systems, is very convenient when your users run an unusually large number of programs. If you rely on this feature too much, however, performance suffers because disk accesses are far slower than are RAM accesses. It’s also important that you have adequate swap space on your system. If the computer runs out of swap space, programs may begin to behave erratically.

Evaluating Swap Space Use

An invaluable tool in checking your system's memory use is **free**. This program displays information on your computer's total memory use. Typically, you just type **free** to use it, but it supports various options that can fine-tune its output. Consult its man page for more information.

Listing 6.1 shows a sample output from **free** on a system with 256MB of RAM. (The total memory reported is less than 256MB because of memory consumed by the kernel and inefficiencies in the *x86* architecture.)

Listing 6.1: Sample Output from **free**

```
$ free
      total        used        free   shared  buffers   cached
Mem:    256452    251600        4852         0     10360    130192
-/+ buffers/cache: 111048    145404
Swap:    515100       1332    513768
```

The Mem line shows the total RAM used by programs, data, buffers, and caches. (All of these values are in kilobytes by default.) Unless you need information on memory used by buffers or caches, this line isn't too useful. The next line, **-/+ buffers/cache**, shows the total RAM used *without* considering buffers and caches. This line can be very informative in evaluating your system's overall RAM requirements and hence in determining when it makes sense to add RAM. Specifically, if the **used** column routinely shows values that approach your total installed RAM (or alternatively, if the **free** column routinely approaches 0), then it's time to add RAM. This information isn't very helpful in planning your swap space use, though.



Real World Scenario

When to Add Swap, When to Add RAM

Swap space exists because hard disks are less expensive than RAM, on a per-megabyte basis. With the price of both falling, however, it's often wise to forgo expanding your swap space in favor of adding extra RAM. RAM is faster than swap space, so all other things being equal, RAM is better.

A general rule of thumb derived from the days of Unix mainframes is that swap space should be 1.5–2 times as large as physical RAM. For instance, a system with 2GB of RAM should have 3–4GB of swap space. If your swap space use regularly exceeds 1.5–2 times your RAM size, your overall system performance will very likely be severely degraded. Adding RAM to such a system will almost certainly improve its performance. It won't hurt to have extra swap space, though, aside from the fact that this reduces the disk space available for programs and data files.

The final line shows swap space use. In the case of Listing 6.1, a total of 515,100KB of swap space is available. Of that, 1,332KB is in use, leaving 513,768KB free. Given the small amount of swap space used, it seems that the system depicted in Listing 6.1 has plenty of swap space, at least assuming this usage is typical.

Adding a Swap File

One method of adding swap space is to create a *swap file*. This is an ordinary disk file that's configured to be used by Linux as swap space. To add a swap file, follow these steps:

1. Create an empty file of the appropriate size. You can do this by copying bytes from `/dev/zero` (a device file that returns bytes containing the value 0) using the `dd` utility. The `dd` program takes parameters of `bs` (block size, in bytes) and `count` (the number of blocks to copy); the total file size is the product of these two values. You specify the input file with `if` and the output file with `of`. For instance, the following command creates a file called `/swap.swp` that's 134,217,728 bytes (128MB) in size:

```
# dd if=/dev/zero of=/swap.swp bs=1024 count=131072
```

2. Use the `mkswap` command to initialize the swap file for use. This command writes data structures to the file to enable Linux to swap memory to disk, but `mkswap` does *not* activate the swap file. For instance, the following command does this job:

```
# mkswap /swap.swp
```



NOTE

Don't try to create a swap file on a network mount. Not only would performance be very poor, but the Network File System (NFS) doesn't support swap files.

3. Use the `swapon` command to begin using the newly initialized swap space:

```
# swapon /swap.swp
```

If you use `free` before and after performing these steps, you should see the total swap space count increase, reflecting the addition of the new swap space. If you want to make your use of this swap file permanent, you must add an entry to `/etc/fstab` (described later in the section "Defining Standard Filesystems"). This entry should resemble the following:

```
/swap.swp swap swap defaults 0 0
```

One key point is to list the complete path to the swap file in the first column, including the leading `/`. Once this entry is added, the system will use the swap file after you reboot. If you want to use all of the swap spaces defined in `/etc/fstab`, type `swapon -a`, which causes Linux to read `/etc/fstab` and activate all the swap partitions defined there.

To deactivate use of swap space, use the `swapoff` command:

```
# swapoff /swap.swp
```

This command may take some time to execute if the swap file has been used much because Linux must move data to other swap areas or to RAM. To disable all swapping, type **swapon -a**, which deactivates all swap spaces—both those listed in `/etc/fstab` and those you’ve added manually.

Adding swap space in the form of a swap file can be a convenient way to add swap space quickly; however, this approach does have certain problems. Most importantly, if you create a large swap file on a partition that’s already been heavily used, it’s likely that the swap file will be *fragmented*; that is, the file’s contents will be spread across multiple groups of sectors on the disk. Fragmentation of disk files slows performance, and this can be a major problem in a swap file. The ability to quickly add a temporary swap file makes this method appealing in many cases, though.

Adding a Swap Partition

Traditionally, Unix and Linux have used *swap partitions* for swap space. These are entire disk partitions devoted to swap space. Most distributions create at least one swap partition during installation. Therefore, chances are good you already have such a partition configured.



If you want to install multiple Linux distributions on one computer, they may share a single swap partition.

What if your existing swap partition is too small, though? You can create a supplementary swap file, as described earlier. Another approach is to create a new swap partition. This procedure works best if you’re adding a hard disk or want to repartition the disk for some other reason. In this case, you’ll be adjusting your partition layout anyway, so you might as well take the opportunity to add new swap space. The basic procedure for doing this is as follows:

1. Clear space for the swap partition. This can be done by deleting existing partitions, by shrinking existing partitions, or by using a previously unused hard disk.
2. Create a new partition. When using an MBR disk, give the new partition a type code of 0x82 (“Linux swap”); with GPT, flag it as a Linux swap partition using whatever method your partitioning software supports. Many OSs (but not Linux) use type codes to help them identify their partitions. MBR type codes 0x82 and 0x83 stand for Linux swap and filesystem partitions, respectively. The main reason to use these codes is to keep other OSs from damaging the Linux partitions.



Solaris for x86 uses the 0x82 partition type code for its own filesystem partitions. This fact can lead to confusion and perhaps even damage, so be careful if your system dual-boots Linux and Solaris.

3. When you're done partitioning or repartitioning, use `mkswap` to prepare the swap partition to be swap space. This operation works just like using `mkswap` on a file, except that you apply it to a partition:

```
# mkswap /dev/sdc6
```

4. Once the swap space has been prepared for use, you can add it manually using the `swapon` command described earlier, but you'll need to specify the swap partition's device rather than a swap file.
5. To use the swap partition permanently, add an entry for it to `/etc/fstab`, as described earlier in reference to swap files.

This procedure glosses over several critically important details concerning partition management. For one thing, when you modify an existing disk's partitions, you may need to adjust the device filenames for Linux filesystems in `/etc/fstab`. You'll have to do this either from an emergency boot or *before* you make the changes to the disk. It is at least as important, if you delete any existing partitions, to back up their contents before you delete the partition, even if you intend to re-create the partition with a smaller size. You may also need to reinstall the LILO or GRUB boot loader if you modify your boot partition. In any event, this procedure will require the use of a disk partitioning tool such as Linux's `fdisk` or a partition-resizing tool such as GNU Parted.

Setting Filesystem Quotas

Just one or two users of a multiuser system can cause serious problems for others by consuming too much disk space. If a single user creates huge files (say, multimedia recordings), those files can prevent other users from creating their own files. To help manage this situation, Linux supports *disk quotas*—limits enforced by the OS on how many files or how much disk space a single user may consume. The Linux quota system supports both quotas for individual users and for Linux groups.

Quotas require support both in the kernel for the filesystem being used and in various user-space utilities. As of the early 2.6.28 kernel, `ext2fs`, `ext3fs`, and `ReiserFS` support quotas, but you must explicitly enable support via the Quota Support kernel option in the filesystem area when recompiling your kernel. Many distributions ship with this support precompiled, so recompiling your kernel may not be necessary, but you should be aware of this option if you do recompile your kernel.

Two general quota support systems are available for Linux. The first was used through the 2.4.x kernels and is referred to as the *quota v1 support*. The second was added with the 2.6.x kernel series and is referred to as the *quota v2 system*. This description applies to the latter system, but the former works in a similar way.

Outside of the kernel, you need support tools to use quotas. For the quota v2 system, this package is usually called `quota`, and it installs a number of utilities, configuration files, SysV startup scripts, and so on.

You must modify your `/etc/fstab` entries for any partitions on which you want to use the quota support. In particular, you must add the `usrquota` filesystem mount option to employ user quotas, and you must add the `grpquota` option to use group quotas. Entries that are so configured resemble the following:

```
/dev/hdc5 /home ext3 usrquota,grpquota 1 1
```

This line activates both user and group quota support for the `/dev/hdc5` partition, which is mounted at `/home`. Of course, you can add other options if you like.



The format of the `/etc/fstab` file is described in more detail in later in this chapter in “Defining Standard Filesystems.”

Depending on your distribution, you may need to configure the quota package’s SysV startup scripts to run when the system boots. Chapter 4, “Managing System Services,” describes SysV startup script management in detail.

After installing software and making configuration file changes, you must activate the systems. The simplest way to do this is to reboot the computer, and this step is necessary if you had to recompile your kernel to add quota support directly into the kernel. If you didn’t do this, though, you should be able to get by with less disruptive measures: using `modprobe` to install the kernel module, if necessary; running the SysV startup script for the quota tools; and remounting the filesystems on which you intend to use quotas by typing `mount -o remount /mount-point`, where `/mount-point` is the mount point in question.

At this point, quota support should be fully active on your computer, but the quotas themselves are not set. You can set the quotas by using `edquota`, which starts the Vi editor (described in Chapter 3, “Managing Processes and Editing Files”) on a temporary configuration file (`/etc/quotatab`) that controls quotas for the user you specify. When you exit from the utility, `edquota` uses the temporary configuration file to write the quota information to low-level disk data structures that control the kernel’s quota mechanisms. For instance, you might type `edquota sally` to edit sally’s quotas. The contents of the editor will show the current quota information:

Quotas for user sally:

```
/dev/hdc5: blocks in use: 3209, limits (soft = 5000, hard = 6500)
          inodes in use: 403, limits (soft = 1000, hard = 1500)
```

The temporary configuration file provides information on both the number of disk blocks in use and the number of inodes in use. (Each file or symbolic link consumes a single inode, so the inode limits are effectively limits on the number of files a user may own. Disk blocks vary in size depending on the filesystem and filesystem creation options, but they typically range from 512 bytes to 8KB.) Changing the use information has no effect, but you can alter the soft and hard limits for both blocks and inodes. The hard limit is the maximum number of blocks or inodes that the user may consume; the kernel will not permit a user to surpass these limits. Soft limits are somewhat less stringent; users may temporarily exceed soft limit

values, but when they do so, the system issues warnings. Soft limits also interact with a grace period; if the soft quota limit is exceeded for longer than the grace period, the kernel begins treating it like a hard limit and refuses to allow the user to create more files. You can set the grace period by using `edquota` with its `-t` option, as in `edquota -t`. Grace periods are set on a per-filesystem basis, rather than a per-user basis.

A couple more quota-related commands are useful. The first is `quotacheck`, which verifies and updates quota information on quota-enabled disks. This command is normally run as part of the quota package's SysV startup script, but you may want to run it periodically (say, once a week) as a cron job. (Chapter 3 describes cron jobs.) Although theoretically not necessary if everything works correctly, `quotacheck` ensures that quota accounting doesn't become inaccurate. The second useful auxiliary quota command is `repquota`, which summarizes the quota information on the filesystem you specify, or on all filesystems if you pass it the `-a` option. This tool can be very helpful in keeping track of disk usage.

Partition Control

In addition to creating partitions and the filesystems they contain, system administrators must be able to control where these low-level filesystems are mounted in the Linux directory tree. If you want to make your changes permanent, you must modify a file called `/etc/fstab`. On a high-performance system, you might also want to link two or more disks together to improve performance or reliability.

Identifying Partitions

If you installed Linux on the system, chances are you told it what partitions to use. If you don't remember what Linux called your partitions at system installation, you can use the `fdisk` program to find out. Pass it the `-l` parameter (that's a lowercase L, not a number 1) and the name of a disk device to obtain a listing of the partitions on that disk:

```
# fdisk -l /dev/hdb
```

```
Disk /dev/hdb: 255 heads, 63 sectors, 1216 cylinders
Units = cylinders of 16065 * 512 bytes
```

Device	Boot	Start	End	Blocks	Id	System
/dev/hdb1		257	1216	7711200	5	Extended
/dev/hdb2		1	192	1542208+	fb	Unknown
/dev/hdb3		193	256	514080	17	Hidden HPFS/ NTFS
/dev/hdb5		257	516	2088418+	6	FAT16
/dev/hdb6	*	517	668	1220908+	7	HPFS/NTFS
/dev/hdb7		669	1216	4401778+	83	Linux

This output shows the device name associated with each partition, the start and end cylinder numbers, the number of 1024-byte blocks in each partition, each partition’s hexadecimal (base 16) ID code, and the partition or OS type associated with that code. The `gdisk` program creates a similar display for GPT disks.

If Linux boots, you can also use the `df` utility (described later in “Using *df*”) to identify the partitions your system is using. This tool won’t identify partitions that aren’t mounted, though, including swap partitions and partitions you simply aren’t using (such as those for non-Linux OSs, unless they’re currently mounted).

Mounting and Unmounting Partitions

Linux provides the `mount` command to *mount* a filesystem to a *mount point*—that is, to make the filesystem available as files and directories in the specified mount point, which is an ordinary directory. The `umount` command reverses this process. (Yes, `umount` is spelled correctly; it’s missing the first *n*.) In practice, using these commands is usually not too difficult, but they support a large number of options.

Syntax and Parameters for *mount*

The syntax for `mount` is as follows:

```
mount [-alrsvw] [-t fstype] [-o options] [device] [mountpoint]
```

Common parameters for `mount` support a number of features, as summarized in Table 6.6. *device* is the device filename associated with the partition or disk device, and *mountpoint* is the directory from which you want to access your files. Both parameters are normally required, but you may omit one of them under certain circumstances, as described shortly.

TABLE 6.6 `mount` Options

Option	Description
-a	This parameter causes <code>mount</code> to mount all the filesystems listed in the <code>/etc/fstab</code> file, which specifies the most-used partitions and devices. The upcoming section “Defining Standard Filesystems” describes this file’s format.
-r	This parameter causes Linux to mount the filesystem read-only, even if it’s normally a read/write filesystem.
-v	As with many commands, this option produces verbose output—the program provides comments on operations as they occur.
-w	This parameter causes Linux to attempt to mount the filesystem for both read and write operations. This is the default for most filesystems, but some experimental drivers default to read-only operation.

TABLE 6.6 mount Options (continued)

Option	Description
-t <i>fstype</i>	Use this parameter to specify the filesystem type. Common filesystem types are ext2 (for ext2fs), ext3 (for ext3fs), ext4 (for ext4fs), reiserfs (for ReiserFS), jfs (for JFS), xfs (for XFS), vfat (for FAT with VFAT long filenames), msdos (for FAT using only short DOS filenames), iso9660 (for most CD-ROM filesystems), udf (for the newer UDF CD-ROM/DVD-ROM filesystem), nfs (for NFS network mounts), smbfs (for SMB/CIFS network shares), and cifs (a newer driver for SMB/CIFS network shares). Linux supports many others. If this parameter is omitted, Linux will attempt to autodetect the filesystem type.
-o	You can add filesystem-specific options using the -o parameter. See Table 6.7.



Linux requires support in the kernel or as a kernel module to mount a filesystem of a given type. If this support is missing, Linux will refuse to mount the filesystem in question.

Table 6.6 isn’t comprehensive; consult the mount man page for some of the more obscure options. The most common applications of mount use few parameters, because Linux generally does a good job of detecting the filesystem type, and the default parameters work reasonably well. For instance, consider this example:

```
# mount /dev/sdb7 /mnt/shared
```

This command mounts the contents of /dev/sdb7 on /mnt/shared, autodetecting the filesystem type and using the default options. Ordinarily, only root may issue a mount command; however, if /etc/fstab specifies the user, users, or owner option, an ordinary user may mount a filesystem using a simplified syntax in which only the device *or* mount point is specified, but not both. For instance, a user might type **mount /mnt/cdrom** to mount a CD-ROM, if /etc/fstab specifies /mnt/cdrom as its mount point and uses the user, users, or owner option.



Many Linux distributions ship with automounter support, which causes the OS to automatically mount removable media when they’re inserted. In GUI environments, a file browser may also open on the inserted disk. In order to eject the disk, the user will need to unmount the filesystem by using **umount**, as described in the upcoming section “Using *umount*,” or by selecting an option in the desktop environment.

When Linux mounts a filesystem, it ordinarily records this fact in `/etc/mtab`. This file has a format similar to that of `/etc/fstab` and is stored in `/etc`, but it's not a configuration file that you should edit. You might examine this file to determine what filesystems are mounted, though. (The `df` command, described in more detail in the section “Using *df*,” is another way to learn what filesystems are mounted.)

Using *mount* Options

When you need to use special parameters, it's usually to add filesystem-specific options. Table 6.7 summarizes the most important filesystem options. Some of these are meaningful only in the `/etc/fstab` file.

TABLE 6.7 Important Filesystem Options for the `mount` Command

Option	Supported Filesystems	Description
<code>defaults</code>	All	Uses the default options for this filesystem. It's used primarily in the <code>/etc/fstab</code> file to ensure that there's an options column in the file.
<code>loop</code>	All	Uses the loopback device for this mount. Enables you to mount a file as if it were a disk partition. For instance, <code>mount -t vfat -o loop image.img /mnt/image</code> mounts the file <code>image.img</code> as if it were a disk.
<code>auto</code> or <code>noauto</code>	All	Mounts or does not mount the filesystem at boot time or when root issues the <code>mount -a</code> command. The default is <code>auto</code> , but <code>noauto</code> is appropriate for removable media. Used in <code>/etc/fstab</code> .
<code>user</code> or <code>nouser</code>	All	Allows or disallows ordinary users to mount the filesystem. The default is <code>nouser</code> , but <code>user</code> is often appropriate for removable media. Used in <code>/etc/fstab</code> . When included in this file, <code>user</code> allows users to type <code>mount /mountpoint</code> , where <code>/mountpoint</code> is the assigned mount point, to mount a disk. Only the user who mounted the filesystem may unmount it.
<code>users</code>	All	Similar to <code>user</code> , except that any user may unmount a filesystem once it's been mounted.

TABLE 6.7 Important Filesystem Options for the mount Command (*continued*)

Option	Supported Filesystems	Description
owner	All	Similar to user, except that the user must own the device file. Some distributions, such as Red Hat, assign ownership of some device files (such as /dev/fd0, for the floppy disk) to the console user, so this can be a helpful option.
remount	All	Changes one or more mount options without explicitly unmounting a partition. To use this option, you issue a mount command on an already-mounted filesystem, but with remount along with any options you want to change. Can be used to enable or disable write access to a partition, for example.
ro	All	Specifies a read-only mount of the filesystem. This is the default for filesystems that include no write access and for some with particularly unreliable write support.
rw	All read/write filesystems	Specifies a read/write mount of the filesystem. This is the default for most read/write filesystems.
uid=value	Most filesystems that don't support Unix-style permissions, such as vfat, hpfs, ntfs, and hfs	Sets the owner of all files. For instance, uid=500 sets the owner to whoever has Linux user ID 500. (Check Linux user IDs in the /etc/passwd file.)
gid=value	Most filesystems that don't support Unix-style permissions, such as vfat, hpfs, ntfs, and hfs	Works like uid=value but sets the group of all files on the filesystem. You can find group IDs in the /etc/group file.
umask=value	Most filesystems that don't support Unix-style permissions, such as vfat, hpfs, ntfs, and hfs	Sets the umask for the permissions on files. value is interpreted in binary as bits to be removed from permissions on files. For instance, umask=027 yields permissions of 750, or rwxr-x---. Used in conjunction with uid=value and gid=value, this option lets you control who can access files on many foreign filesystems.

TABLE 6.7 Important Filesystem Options for the mount Command (*continued*)

Option	Supported Filesystems	Description
conv= <i>code</i>	Most filesystems used on Microsoft and Apple OSs: msdos, umsdos, vfat, hpfs, and hfs	If <i>code</i> is b or binary, Linux doesn't modify the files' contents. If <i>code</i> is t or text, Linux autoconverts files between Linux-style and DOS- or Macintosh-style end-of-line characters. If <i>code</i> is a or auto, Linux applies the conversion unless the file is a known binary file format. It's usually best to leave this at its default value of binary because file conversions can cause serious problems for some applications and file types.
norock	iso9660	Disables Rock Ridge extensions for ISO-9660 CD-ROMs.
nojoliet	iso9660	Disables Joliet extensions for ISO-9660 CD-ROMs.

Some filesystems support additional options that aren't described here. The mount man page covers some of these, but you may need to look to the filesystem's documentation for some filesystems and options. This documentation may appear in `/usr/src/linux/Documentation/filesystems` or `/usr/src/linux/fs/fsname`, where *fsname* is the name of the filesystem.

Using *umount*

The `umount` command is simpler than `mount`. The basic `umount` syntax is as follows:

```
umount [-afnr] [-t fstype] [device | mountpoint]
```

Most of these parameters have similar meanings to their meanings in `mount`, but some differences deserve mention, as summarized in Table 6.8.


TABLE 6.8 `umount` Options that Differ from `mount` Options

Option	Description
-a	Rather than unmount partitions listed in <code>/etc/fstab</code> , this parameter causes the system to attempt to unmount all the partitions listed in <code>/etc/mtab</code> , the file that holds information on mounted filesystems. On a normally running system, this operation is likely to succeed only partly because it won't be able to unmount some key filesystems, such as the root partition.

TABLE 6.8 umount Options that Differ from mount Options *(continued)*

Option	Description
-f	With this option you can tell Linux to force an unmount operation that might otherwise fail. This feature is sometimes helpful when unmounting NFS mounts shared by servers that have become unreachable.
-r	This option tells umount that if it can't unmount a filesystem, it should attempt to remount it in read-only mode.
-t <i>fstype</i>	This option tells the system to unmount only partitions of the specified type. You can list multiple filesystem types by separating them with commas.
<i>device</i> and <i>mountpoint</i>	You need to specify only the <i>device</i> or only the <i>mountpoint</i> , not both.

As with `mount`, normal users cannot ordinarily use `umount`. The exception is if the partition or device is listed in `/etc/fstab` and specifies the `user`, `users`, or `owner` option, in which case normal users can unmount the device. (In the case of `user`, only the user who mounted the partition may unmount it; and in the case of `owner`, the user issuing the command must also own the device file, as with `mount`.) These options are most useful for removable-media devices.



WARNING

Be cautious when removing floppy disks, USB flash drives, and certain other removable disks. Linux caches accesses to all disks, which means that data may not be written to the disk until some time after a write command. Because of this, it's possible to corrupt a removable disk by ejecting it, even when the drive isn't active. You must *always* issue a `umount` command before ejecting a mounted disk. This isn't an issue for some removable media because Linux can lock their eject mechanisms, preventing this sort of problem.

Using Network Filesystems

Although they aren't local disk partitions, network filesystems can be mounted using the same commands used to mount local disk partitions and removable disks. They do possess certain unique features, though. Two network filesystems are most common in Linux: NFS, which is commonly used among Unix and Unix-like operating systems, and the Server Message Block/Common Internet File System (SMB/CIFS), which is most strongly associated with Windows systems, although the Samba server for Linux can also deliver SMB/CIFS shares.



Chapter 11 describes configuring NFS and Samba servers in Linux. This chapter covers the client side.

Accessing SMB/CIFS Shares

Microsoft Windows uses SMB/CIFS for file and printer sharing. Linux includes tools that provide the ability to interact with Windows systems that use SMB/CIFS. The main package for this is called Samba, and it comes with all major Linux distributions. Samba includes two major client programs: `smbclient` and `smbmount`. The `smbclient` program is modeled after the `ftp` client program, which is described in Chapter 11. The `smbmount` utility actually mounts the share in the Linux directory tree. The standard Linux `mount` command can also mount SMB/CIFS shares.

To use `smbmount`, type `smbmount //server/share /mount/point`, where *server* and *share* are the name of the server and the share you want to access, respectively, and */mount/point* is the local mount point you want to use. You'll be asked to provide a password. (By default, `smbmount` passes your login name as your username.) You can then use standard Linux file-access commands on the share. When you're done, you can use `smbumount` to unmount the share.

One drawback to `smbmount` is that it assigns Linux ownership of all files on the remote server to the user who ran the command, unless you use the `-o uid=UID` option, which sets ownership to the user whose user ID is *UID*. You might also need to use the `-o username=name` option to set the username used to access the shares.



For ordinary users to run `smbmount` and `smbumount`, the `smbmnt` and `smbumount` programs must have their SUID bits set, which allows ordinary users to run programs with root privileges. (`smbmnt` is a helper program to `smbmount`.) If this isn't the case when Samba is installed, type `chmod a+s /usr/bin/smbmnt /usr/bin/smbumount` as root. Thereafter, ordinary users will be able to use these programs, but they'll need to own the mount points they use.

Another way to mount SMB/CIFS shares is via the standard Linux `mount` command. This requires you to pass a filesystem type of either `smbfs` or `cifs` with the `-t` parameter, along with the server and share name rather than a local Linux device filename:

```
# mount -t smbfs //apollo/hschmidt /mnt/a17
```

The `smbfs` filesystem type code is older than `cifs` and is being phased out in favor of `cifs`, which adds support for Unix-specific extensions to SMB/CIFS. These extensions enable `cifs` to provide limited support for ownership, permissions, symbolic links, and other Linux-style filesystem information. These features are important only when the server supports them, though. Windows servers do not do so, although Samba does. Thus, using `cifs` may make sense when mounting shares from a Samba server. On the other hand,

some older clients, such as Windows 9x/Me, lack support for the protocols required by the `cifs` driver. Therefore, if you want to mount shares from such systems, you *must* use `smbfs` rather than `cifs`.

Accessing NFS Exports

Like SMB/CIFS, Sun’s NFS is a file sharing protocol, but it was designed with the needs of Unix systems in mind. NFS includes Unix features, such as support for owners, groups, and permission strings that aren’t well supported by SMB/CIFS. (The Unix CIFS extensions narrow this gap, but NFS tends to handle these features better.) Because Linux conforms closely to the Unix model, NFS is the preferred method for file sharing between Linux systems.

In Linux, client access to NFS exports is tightly integrated into normal Linux file-access utilities. Specifically, you use the `mount` command to mount the NFS exports, and you can then access files stored on the NFS server as if they were ordinary files. To do so, you provide `mount` with a server hostname or IP address and a path to the directory on the server you want to access, rather than a device filename. For instance, you might issue commands like the following:

```
# mount apollo:/home/hschmidt /mnt/a17
# ls -l /mnt/a17
total 152
-rwxr-xr-x 1 rodsmith users 152576 Mar 29 13:01 drrock.wpd
drwxr-xr-x 1 rodsmith users 512 Apr 2 2000 geology
# cp /mnt/a17/drrock.wpd ./
# umount /mnt/a17
```

It’s important to note that you aren’t required to enter a password when you access NFS exports. An NFS server allows a specified set of clients to access the exported directories in a more-or-less unrestricted manner; the server relies on the client’s security policies to prevent abuses.

Using *df*

If you need information on disk space used on an entire partition, the `df` command does the job. This command summarizes total, used, and available disk space. You can provide options to `df` to vary the data it produces, as summarized in Table 6.9.

TABLE 6.9 `df` Options

Option	Option Abbreviation	Description
<code>--human-readable</code>	<code>-h</code>	Normally, <code>df</code> provides output in 1024-byte blocks. This option makes it provide listings in labeled units of kilobytes (k), megabytes (M), or gigabytes (G) instead.

TABLE 6.9 df Options (continued)

Option	Option Abbreviation	Description
--inodes	-i	By default, df displays disk space used, but this option causes df to display information on the consumption of inodes. Some filesystems, such as ext2fs, have a fixed number of inodes when formatted. Others, such as FAT and ReiserFS, don't, so this information is spurious or meaningless with these filesystems.
--local	-l	This option causes df to ignore network filesystems.
--print-type	-T	This option causes df to display the filesystem type code along with other information.

You can type **df** alone or in combination with options to obtain information on your system's mounted partitions. If you want information on just one partition, you can add either the device on which it resides or any file or directory on the filesystem to restrict df's output to that one partition. In action, df works like this:

```
# df -hT
Filesystem      Type  Size  Used Avail Use% Mounted on
/dev/hda9       ext2  2.0G  1.8G   96M  95% /
/dev/hdb5       vfat  2.0G  1.4G  564M  72% /mnt/windows
speaker:/home   nfs   4.5G  2.2G  2.3G  49% /mnt/speaker/home
/dev/hdb7       reiserfs 4.2G  1.9G  2.3G  45% /home
```

The df command is extremely useful in discovering how much free space is available on a disk and how well distributed across partitions your files are.



Linux's ext2, ext3, and ext4 filesystems normally reserve about 5 percent of their available space for root. The intent is that if users come close to filling the disk, there'll be enough space for the system administrator to log in and perform basic maintenance to correct problems. If a critical filesystem were to fill completely, root might not be able to log in.

Defining Standard Filesystems

The `/etc/fstab` file controls how Linux provides access to disk partitions and removable media devices. (The filename `fstab` is an abbreviation for “filesystem table.”) This file consists

of a series of lines, each of which contains six fields that are separated by one or more spaces or tabs. A line that begins with a hash mark (#) is a comment and is ignored. Listing 6.2 shows a sample /etc/fstab file.

Listing 6.2: Sample /etc/fstab File

#device	mount point	filesystem	options	dump	fsck
/dev/hda1	/	ext3	defaults	1	1
LABEL=/home	/home	reiserfs	defaults	0	0
/dev/hdb5	/windows	vfat	uid=500,umask=0	0	0
/dev/hdc	/mnt/cdrom	iso9660	user,noauto	0	0
/dev/fd0	/mnt/floppy	auto	user,noauto	0	0
server:/home	/other/home	nfs	user,exec	0	0
//winsrv/shr	/other/win	smbfs	user,credentials=/etc/creds	0	0
/dev/hda4	swap	swap	defaults	0	0

The meaning of each field in this file is as follows:

Device The first column specifies the mount device. These are usually device filenames that reference hard disks, floppy drives, and so on. Alternatively, a label or globally unique identifier (GUID) may be specified, as in LABEL=/home or GUID=2F9FDF9-121F-031C-C08B-75B32AFBB108. (Most filesystems support labels and GUID numbers; using them can help keep the system bootable in case partition table changes alter partition numbers.) It's also possible to list a network drive, as in server:/home, which is the /home export on the computer called server.

Mount point The second column specifies the mount point. This should usually be an empty directory in another filesystem. The root (/) filesystem is an exception. So is swap space, which is indicated by an entry of swap.

Filesystem type The filesystem type code is the same as the type code used to mount a filesystem with the mount command. You can use just about any filesystem type code you can use directly with the mount command. A filesystem type code of auto causes the kernel to autodetect the filesystem type, which can be a convenient option for removable media devices. Autodetection doesn't work with all filesystems, though.

Mount options You can specify mount options, separated by commas, in the fourth field. Table 6.7 summarizes common options, although more are available. For instance, uid=500,umask=0 for /windows in Listing 6.2 sets the user ID (owner) of all files to 500 and sets the umask to 0. Type **man mount** or consult filesystem-specific documentation to learn more.

dump operation The next-to-last field contains a 1 if the dump utility should back up a partition, or a 0 if it should not. If you never use the dump backup program, this option is essentially meaningless. The dump program is a common backup tool, but it's by no means the only one.

Filesystem check order The final column specifies the order in which the boot-time filesystem check occurs. A 0 means that `fsck` should *not* check a filesystem. Higher numbers represent the check order. The root partition should have a value of 1, and all others that should be checked should have a value of 2. Some filesystems, such as ReiserFS, should not be automatically checked and so should have values of 0.

If you add a new hard disk or have to repartition the one you have, you'll probably need to modify `/etc/fstab`. You might also need to edit it to alter some of its options. For instance, setting the user ID or umask on Windows partitions mounted in Linux may be necessary to let ordinary users write to the partition.

The `credentials` option for the `/other/win` mount point in Listing 6.2 deserves greater elaboration. Ordinarily, most SMB/CIFS shares require a username and password as a means of access control. Although you can use the `username=name` and `password=pass` options to `smbfs` or `cifs`, these options are undesirable, particularly in `/etc/fstab`, because they leave the password vulnerable to discovery—anybody who can read `/etc/fstab` can read the password. The `credentials=file` option provides an alternative—you can use it to point Linux at a file that holds the username and password. This file has labeled lines:

```
username=hschmidt
password=yiW7t9Td
```

Of course, the file you specify (`/etc/creds` in Listing 6.2) must be well protected—it must be readable only to root and perhaps to the user whose share it describes.

Using RAID

Two problems with traditional disk subsystems plague high-performance computers such as midsize and large servers:

Reliability Although modern hard disks are reliable enough for most uses, the consequences of disk failure on truly mission-critical systems can be catastrophic. If the reliability of disk storage can be improved, it should be.

Speed Systems that transfer large amounts of data often run into the speed limitations of modern hard disks.

Both of these problems can be overcome, or at least minimized, by using a technology known as *redundant array of independent disks (RAID)*. Several different forms of RAID exist, and using them requires additional Linux configuration.

Forms of RAID

RAID uses multiple disks and special drivers or controllers. Linux supports several varieties of RAID, each with its own features and priorities:

Linear (append) The linear approach is very simple: it enables you to combine partitions from multiple disks into a single large virtual partition. It's more useful for creating partitions larger than your individual disks support than for anything else; it provides no reliability or speed

benefits. Total capacity is identical to using the drives in a conventional configuration. Logical volume management (LVM), described shortly, provides similar features.

RAID 0 (striping) The RAID 0 approach is similar to linear mode, but it interleaves data intended for each physical disk—that is, the combined logical partition consists of small strips from each component disk. The result is improved performance, because disk accesses are spread across multiple physical disks. Reliability is not improved, however, and could actually be degraded compared to using a single larger disk, because a failure of *any* disk in the array will cause data loss. Total capacity is identical to using the drives in a conventional configuration. Linux’s LVM subsystem can be configured to employ a similar striping feature without using RAID.

RAID 1 (mirroring) A RAID 1 array uses one disk to exactly duplicate the data on another disk—when you write data to the first disk, the data is actually written to both disks. This provides redundancy that can protect against drive failures, but it slows performance, at least when it’s implemented in the OS. (Some hardware RAID controllers can perform this task without a performance hit.) Total capacity is the same as having a single drive—the extra drives provide improved reliability, not capacity per se.

RAID 4/5/6 These RAID types combine the features of RAID 0 and RAID 1: they spread data across multiple disks and provide redundancy. They do this by using parity bits, which can be used to regenerate data should a single drive stop functioning. RAID 4 stores the parity bits on a single drive, whereas RAID 5 stores them on all the drives. In both cases, a set of N identical drives provides a capacity equal to $N-1$ drives for RAID 4 or 5. RAID 6 protects against the failure of two disks rather than just one, but with the disadvantage that N disks provide the capacity of $N-2$ disks.

RAID versions of 1 and above support *hot standby*—a feature that enables an extra drive to be automatically activated and used should one of the main drives fail. This feature requires adding one more drive to the array, above and beyond the requirements described earlier.

Designing a RAID Array

RAID configuration requires that you decide how to combine multiple partitions to best effect. In theory, you can combine just about any partitions; however, some techniques will help you get the most from your RAID array:

Ensure your computer is adequate. Old computers may lack the internal data-processing capacity to make effective use of a RAID array of modern disks. Ideally, the disk controller circuitry should be built into the motherboard’s chipset, which can improve its throughput.

Place disks on different controllers. For best performance, use different disk controllers or host adapters for your disks. This advice is less important for SCSI or SATA than for PATA; PATA support for multiple simultaneous transfers on a single controller is very limited, so you shouldn’t attempt to combine the master and slave devices on one cable into a single RAID array.

Use hardware RAID. Some disk controllers support hardware RAID. These devices can provide superior performance, particularly for RAID 1 and above. Unfortunately, identifying these controllers can be tricky—many claim to support RAID, but they really provide a few minimal hooks and Windows drivers. Such devices present no advantages in Linux over conventional controllers. If you use a hardware RAID controller, consult its documentation, and the documentation for its Linux drivers, for information on its use; the upcoming section “Configuring Linux RAID” does *not* apply to such controllers.

Use disks of similar performance. You should use disks that are as similar as possible in performance and capacity—ideally, all the disks in an array should be the same model. If performance varies wildly between disks, you’d probably be better off simply using the faster drive to hold critical filesystems than trying to use a RAID array, at least if your goal is improved disk performance.

Use identically sized partitions. Linux’s RAID configuration combines partitions together. This works best when the partitions are as close as possible in size. If you try to combine partitions of different sizes, the “extra” space in the larger partition will be wasted.

Configure the system to boot using RAID. Unless you use a hardware RAID controller, your computer’s BIOS won’t understand your RAID configuration. Because the BIOS must read the kernel, you must either place your kernel on a non-RAID partition or use RAID 1 for your kernel’s partition (which enables you to refer to an underlying Linux partition in your boot loader). If you want a wholly RAID computer, you can create a separate `/boot` partition as RAID 1 and use RAID 0 or RAID 4/5/6 for your remaining partitions.

You can mix and match RAID types on a single Linux RAID array and even use some non-RAID partitions. (In the latter case, you must either create identically sized non-RAID partitions on all the array’s disks or use disks of unequal size, filling the extra space in the larger disks with non-RAID partitions.)

Configuring Linux RAID

To use RAID, you must compile support into your kernel. This support is provided by default by most distributions, but if you need to activate it, look in the Device Drivers ➤ Multi-Device Support (RAID and LVM) section of the kernel.

In addition to kernel support, using RAID requires one of two software packages: `raidtools` or `mdadm`. Both tools ship with most distributions. The tools differ in their approaches: `raidtools` uses a configuration file, `/etc/raidtab`, to define RAID arrays, whereas `mdadm` is a command-line program in which you can create RAID arrays interactively. This section emphasizes the use of `raidtools`. Whichever program you use, you should use `fdisk` (described earlier in “Using `fdisk` to Create Partitions”) to convert the partitions’ type codes to `0xFD`, using the `t` command in `fdisk`; or you should set the RAID flag using GNU Parted’s `set` command. The `0xFD` MBR type code (or equivalent GPT type code for GPT disks) identifies Linux RAID partitions. Upon boot, Linux will search these partitions for RAID information and should combine them.

To actually define your RAID configuration using `raidtools`, you use a file called `/etc/raidtab`. A simple RAID 1 configuration looks like this:

```
raiddev /dev/md0
    raid-level          1
    nr-raid-disks       2
    persistent-superblock 1
    nr-spare-disks      1
    device              /dev/sda1
    raid-disk            0
    device              /dev/sdb1
    raid-disk            1
    device              /dev/sdc1
    spare-disk          0
```

This configuration creates a RAID 1 (`raid-level`) device that will subsequently be accessed as `/dev/md0` (`raiddev`). This configuration uses two disks (`nr-raid-disks`) and enables a persistent superblock, which is how Linux stores its RAID information within each RAID partition. The `nr-spare-disks` line defines the number of hot standby disks that are held in reserve—if another disk fails, a spare disk may be automatically called up by the RAID tools as a replacement. (Note that the spare disks, if used, are *not* counted among the RAID disks on the `nr-raid-disks` line.) The following pairs of lines define the partitions that are to be used in the RAID array. The main disks are identified by their conventional device filenames (`device`) and given numbers starting with 0 (`raid-disk`). If a spare disk is used, it's identified and numbered using the `spare-disk` directive as well.

A RAID 5 configuration looks much the same but adds a few lines:

```
raiddev /dev/md1
    raid-level          5
    nr-raid-disks       3
    nr-spare-disks      0
    persistent-superblock 1
    parity-algorithm    left-symmetric
    chunk-size          32
    device              /dev/sda2
    raid-disk            0
    device              /dev/sdb2
    raid-disk            1
    device              /dev/sdc2
    raid-disk            2
```

The first main addition to this configuration is `parity-algorithm`, which sets how the parity bits should be computed. Possible options are `left-symmetric`, `right-symmetric`,

left-asymmetric, and right-asymmetric. The first of these options usually provides the best performance. The `chunk-size` option sets the size of the stripes used in the array, in kilobytes. This value must be a power of 2. Typical values range from 4 to 128. The best value depends on your hardware, so if you must have the best performance, you'll have to experiment; otherwise, a value of 32 is reasonable.

Once you've created your `/etc/raidtab` file, you must initialize the system by using `mkraid`, which takes one or more RAID device filenames as options:

```
# mkraid /dev/md0 /dev/md1
```

This command reads `/etc/raidtab` and initializes the specified devices using the settings in that file. If `mkraid` detects data on the partitions, it may complain; to force it to proceed without complaint, include the `-f` option. Once this is done, you can treat these devices as if they were ordinary disk partitions, creating filesystems and storing files on them. You can even refer to them in `/etc/fstab` to mount them automatically when the system boots.



The `mkraid` command destroys all data on the partitions in question. You should run it only on *new* RAID arrays, and you should double- and triple-check your `/etc/raidtab` file to be sure you haven't inadvertently specified non-RAID disks for inclusion in an array.

Using LVM

A second type of advanced volume tool, beyond RAID, is LVM. RAID is designed to improve the speed and reliability of disks, but LVM is designed to increase the flexibility of partition access by making it easier to create, delete, and resize low-level filesystems. You can use LVM with or without using RAID. To employ LVM, you must set up three different levels of data structures: physical volumes, volume groups, and logical volumes.

Understanding LVM

Partitions are defined in a crude and simple way compared to files on a filesystem, so changing your partition layout is potentially dangerous, slow, and awkward. For instance, suppose a disk has four equally sized partitions, numbered 1–4. Consolidating partitions 2 and 4, if that becomes desirable, is an awkward task, because you'll need to move all the data on partition 3 to create a single contiguous space for the newly consolidated partition. Backing up the contents of partition 2 or 4 and then restoring the partition may also be necessary.

LVM simplifies matters by creating *logical volumes* as substitutes for partitions. A logical volume is handled in a filesystem-like data structure known as a *volume group*. If you want to consolidate logical volumes 2 and 4, you don't need to touch logical volumes 1 and 3 (although you may still need to move data between the volumes that are to be consolidated). You can resize logical volumes as desired, without regard to how they're physically laid out on the disk.

LVM also enables you to create a volume group from multiple partitions (known as *physical volumes* in LVM-speak), even across physical disks. This ability, which is similar to linear RAID, can help you create very large filesystems—larger than any one disk supports. You can even stripe the volume group across multiple physical volumes, which can improve performance in a manner reminiscent of RAID 0.

Overall, LVM is a very useful tool on systems whose storage needs are uncertain or likely to change. You can easily add disk space or rearrange how it's allocated. LVM adds complexity, though, and if you create volume groups that span multiple disks, your data will become more vulnerable—if one disk fails, you may no longer be able to access any data from the volume group, even if it was actually stored on the good disk. You should also keep in mind that resizing a logical volume is useful only to the extent that you can resize the underlying filesystem. You can grow or shrink ReiserFS, ext2, ext3, and ext4 filesystems; but JFS and XFS may only be grown, not shrunk. Swap space is best handled by using `swapon` to disable it, resizing the logical volume, using `mkswap` to create a fresh swap area, and then reactivating it with `swapon`.

Ordinarily, a Linux system that uses LVM will still have at least one non-LVM partition. This partition holds the kernel so that your GRUB or LILO boot loader can read it. Typically, `/boot` is set aside for this purpose, although you can put the entire root (`/`) filesystem in a conventional partition, if you prefer. (In fact, you can use any mixture of conventional partitions and logical volumes that you like, so long as you can get the computer booted in some way.)

Defining Physical Volumes

To implement an LVM, you begin by preparing your physical volumes—that is, your partitions. You should mark MBR partitions with the 0x8E partition type code using `fdisk`. If you use GNU Parted with MBR or GPT disks, set the LVM flag with the `set` command.

With the appropriate partition type code set, you can use the `pvccreate` command to set up the low-level physical volume data structures within a partition:

```
# pvccreate /dev/sda4
```

You can repeat this step with multiple partitions, if necessary.

Several additional commands, such as `pvdsize`, `pvrsize`, and `pvsplit`, enable you to perform further physical volume manipulations or to obtain data on your physical volumes.

Defining Volume Groups

With one or more physical volumes defined, you can now create a volume group with the `vgcreate` command:

```
# vgcreate my_group /dev/sda4 /dev/sdb1
```

This command creates a volume group from `/dev/sda4` and `/dev/sdb1` and gives it the name `my_group`. If you already have a volume group and you want to extend it (say, with a new physical disk you've just added to the computer), you can do so with the `vgextend` command:

```
# vgextend my_group /dev/sdc1
```

This partition must have already been initialized with `pvccreate`. The `pvccreate` and `pvextend` commands both support a number of options; consult their man pages for details.

Several additional commands, such as `vgremove`, `vgrename`, `vgdisplay`, `vgscan`, and `vgsplit`, enable you to view data about or manipulate your volume group. Consult these tools' man pages for more information.

Defining Logical Volumes

With your volume group created, you can begin creating logical volumes within it. This is done with the `lvcreate` command:

```
# lvcreate -i 3 -L 100G -n home my_group
```

This example creates a 100GB (`-L 100G`) logical volume called `home` (`-n home`) within the `my_group` volume group. This example is striped across three physical volumes (`-i 3`) for improved performance. You could omit the `-i 3` option, but as a practical matter, you must specify the other options. The result of this command is a new Linux device file, `/dev/mapper/my_group-home`, which points to 100GB of disk space in the volume group. This device file may also be accessed as `/dev/my_group/home`, if you prefer.

Chances are you'll want to run `lvcreate` multiple times to create several logical volumes—perhaps one each for `/home`, `/usr`, `/usr/local`, `/var`, and a few other filesystems that you might ordinarily put on separate partitions. If you use XFS or JFS, you may want to stop short of filling your volume group; that way, you'll have unallocated space you can use to expand any logical volume if the need arises. With ReiserFS, `ext2fs`, `ext3fs`, or `ext4fs`, you can shrink an underutilized volume if you must make room for expanding another logical volume.

The `lvcreate` man page describes many more options for this command, but for the most part they're very technical. There are also additional logical volume information and management commands, such as `lvdisplay`, `lvscan`, `lvresize`, and `lvremove`. Consult their man pages for more information.

Working with Logical Volumes

Once your logical volumes are created, you can use them as if they were partitions: use their device filenames in place of partitions' device filenames in commands such as `mkfs` and `mount`, and place references to these device filenames in `/etc/fstab`. A logical volume can hold either a filesystem or swap space.

Non-Linux OSs can't access Linux LVMs, so you shouldn't use them to hold shared data on a computer that boots multiple OSs. There's also little point in creating physical volumes on removable disks.

Summary

Linux uses a unified filesystem, which means it doesn't use drive letters as Windows does. Instead, partitions are mounted within a single directory structure, starting at the root (`/`) partition. You can create filesystems on partitions or removable disks, mount them, store

files on them, and back them up individually or across partitions. You can mount partitions temporarily or create entries in `/etc/fstab` to make changes permanent, as you see fit. You might also want to create a RAID array or LVM configuration, which can improve reliability, speed, or flexibility.

Exam Essentials

Summarize how Linux’s filesystem (that is, its directory tree) is structured. Linux’s directory tree begins with the root (`/`) directory, which holds mostly other directories. Specific directories may hold specific types of information, such as user files in `/home` and configuration files in `/etc`. Some of these subdirectories and their subdirectories may in fact be separate partitions, which helps isolate data in the event of filesystem corruption.

Describe Linux’s partitioning needs. Linux requires a single root partition and generally uses a separate swap partition. Additional partitions, corresponding to directories such as `/boot`, `/home`, and `/var`, are desirable on some systems but aren’t usually required.

Describe why you might pick particular filesystems for Linux installation. Ext3fs is a popular choice and generally a good one. ReiserFS, XFS, and JFS are also good choices on distributions that support them, but many don’t. The older ext2fs can be a good choice for small partitions but is better avoided for large partitions. The new ext4fs offers the advantages of many other filesystems.

Explain the operation of the mount command. In its basic form, `mount` takes a device filename and directory and ties the two together so that files on the device may be accessed in the specified directory. A number of parameters and options can modify its function or how it treats the filesystem that it mounts.

Identify when swap space needs to be increased. The output of the `free` command shows how much memory Linux is using—both RAM and swap space. When the amount of used swap space approaches available swap space, it’s necessary to increase swap space or RAM.

Explain how Linux knows what partitions to mount when it boots. Linux looks to the `/etc/fstab` file for information on the filesystems it should mount automatically (and perhaps some that it shouldn’t mount automatically but that should be available for users to mount manually).

Know how to create a new filesystem on a disk or partition. The `mkfs` program creates new filesystems on removable media drives or hard disk partitions. This program is actually a front-end to programs that do the actual work, such as `mke2fs` (aka `mkfs.ext2` and `mkfs.ext3`) for ext2fs and ext3fs.

Describe how to check a filesystem for errors. The `fsck` program checks a filesystem’s internal consistency. Like `mkfs`, it’s a front-end to filesystem-specific programs, such as `e2fsck` (aka `fsck.ext2` and `fsck.ext3`) for ext2fs and ext3fs.

Describe the purpose of a RAID array. A RAID array may be used to increase disk speed, disk reliability, or both. The array uses multiple disks to work around individual disk speed limitations or to store duplicate copies of (or checksums for) data.

Describe the purpose of LVM. LVM substitutes logical volumes for partitions, which improves flexibility. Logical volumes can be resized much more easily than can partitions, and this simplifies system reconfiguration in the future. LVM can also improve disk performance by striping together multiple disks in a manner similar to that of RAID 0.

Review Questions

1. Typing **fdisk -l /dev/hda** on an x86 Linux computer produces a listing of four partitions: /dev/hda1, /dev/hda2, /dev/hda5, and /dev/hda6. Which of the following is true?
 - A. The disk contains two primary partitions and two extended partitions.
 - B. Either /dev/hda1 or /dev/hda2 is an extended partition.
 - C. The partition table is corrupted; there should be a /dev/hda3 and a /dev/hda4 before /dev/hda5.
 - D. If you add a /dev/hda3 with **fdisk**, /dev/hda5 will become /dev/hda6, and /dev/hda6 will become /dev/hda7.
2. Which of the following pieces of information can **df** *not* report?
 - A. How long the filesystem has been mounted
 - B. The number of inodes used on an ext3fs partition
 - C. The filesystem type of a partition
 - D. The percentage of available disk space used on a partition
3. A new Linux administrator plans to create a system with separate /home, /usr/local, and /etc partitions. Which of the following best describes this configuration?
 - A. The system won't boot because /etc contains configuration files necessary to mount nonroot partitions.
 - B. The system will boot, but /usr/local won't be available because mounted partitions must be mounted directly off their parent partition, not in a subdirectory.
 - C. The system will boot only if the /home partition is on a separate physical disk from the /usr/local partition.
 - D. The system will boot and operate correctly, provided each partition is large enough for its intended use.
4. What mount point should you associate with swap partitions?
 - A. /
 - B. /swap
 - C. /boot
 - D. None
5. You run Linux's **fdisk** and modify your partition layout. Before exiting from the program, though, you realize that you've been working on the wrong disk. What can you do to correct this problem?
 - A. Nothing; the damage is done, so you'll have to recover data from a backup.
 - B. Type **w** to exit from **fdisk** without saving changes to disk.
 - C. Type **q** to exit from **fdisk** without saving changes to disk.
 - D. Type **u** repeatedly to undo the operations you've made in error.

6. What does the following command accomplish?
`# mkfs -V -t ext2 /dev/sda4`
- A. It sets the partition table type code for /dev/sda4 to ext2.
 - B. It converts a FAT partition into an ext2fs partition without damaging the partition's existing files.
 - C. It creates a new ext2 filesystem on /dev/sda4, overwriting any existing filesystem and data.
 - D. Nothing; the -V option isn't valid, so it causes mkfs to abort its operation.
7. You want to allow Linux users running OpenOffice.org to directly edit files stored on a Windows 2000 SMB/CIFS file server. Which of the following would you use to enable this?
- A. Linux's standard NFS file sharing support
 - B. An FTP server running on the Windows system
 - C. The Linux `smbclient` program
 - D. The Linux `smbmount` program
8. What is wrong with the following /etc/fstab file entry? (Choose all that apply.)
`/dev/hda8 nfs default 0 0`
- A. The entry is missing a mount-point specification.
 - B. All /etc/fstab fields should be separated by commas.
 - C. The `default` option may be used only with ext2 filesystems.
 - D. /dev/hda8 is a disk partition, but `nfs` indicates a network filesystem.
9. Where may a swap file be located?
- A. Only on the root (/) Linux filesystem
 - B. On local read/write Linux filesystems
 - C. On NFS or ext2 filesystems
 - D. On any partition with more than 512MB of free disk space
10. In which of the following situations would it be *most* reasonable to create a new swap partition?
- A. Your heavily used server is nearly out of swap space and needs no routine maintenance.
 - B. A workstation user has been using memory-hungry programs that exceed memory capacity and needs a quick fix.
 - C. You're adding a new hard disk to a multiuser system and expect several new users in the next month or so.
 - D. A system has been experiencing slow performance because of excessive swapping.

11. You've added a new disk to a computer that already uses LVM. You've partitioned the new disk and used `pvccreate` on its one partition in preparation for adding it to your existing LVM. What command should you use to add this partition (`/dev/sdb1`) to your existing `biglvm` volume group?
- A. `pvdiskdisplay /dev/sdb1`
 - B. `vgcreate biglvm /dev/sdb1`
 - C. `vgextend biglvm /dev/sdb1`
 - D. `lvextend biglvm /dev/sdb1`
12. Which of the following is a GUI tool that supports resizing several filesystems, including FAT, ext2fs, and ReiserFS?
- A. QTParted
 - B. GNU Parted
 - C. Part
 - D. cfdisk
13. Which of the following options is used with `fsck` to force it to use a particular filesystem type?
- A. `-A`
 - B. `-N`
 - C. `-t`
 - D. `-C`
14. Which of the following utilities would create the following display?
- | | total | used | free | shared | buffers | cached |
|--------------------|--------|--------|--------|--------|---------|--------|
| Mem: | 256452 | 251600 | 4852 | 0 | 10360 | 130192 |
| -/+ buffers/cache: | 111048 | 145404 | | | | |
| Swap: | 515100 | 1332 | 513768 | | | |
- A. `mt`
 - B. `df`
 - C. `swapon`
 - D. `free`
15. What will be the result of the `root` user running the following command?
- ```
mount /dev/sdc5 /home2
```
- A. The contents of `/home2` will be mounted on `/dev/sdc5` with the default filesystem used and a prompt for options will appear.
  - B. The contents of `/home2` will be mounted on `/dev/sdc5` with the filesystem type auto-detected and default options used.
  - C. The contents of `/dev/sdc5` will be mounted on `/home2` with the filesystem type auto-detected and default options used.
  - D. The contents of `/dev/sdc5` will be mounted on `/home2` with the default filesystem used and a prompt for options will appear.

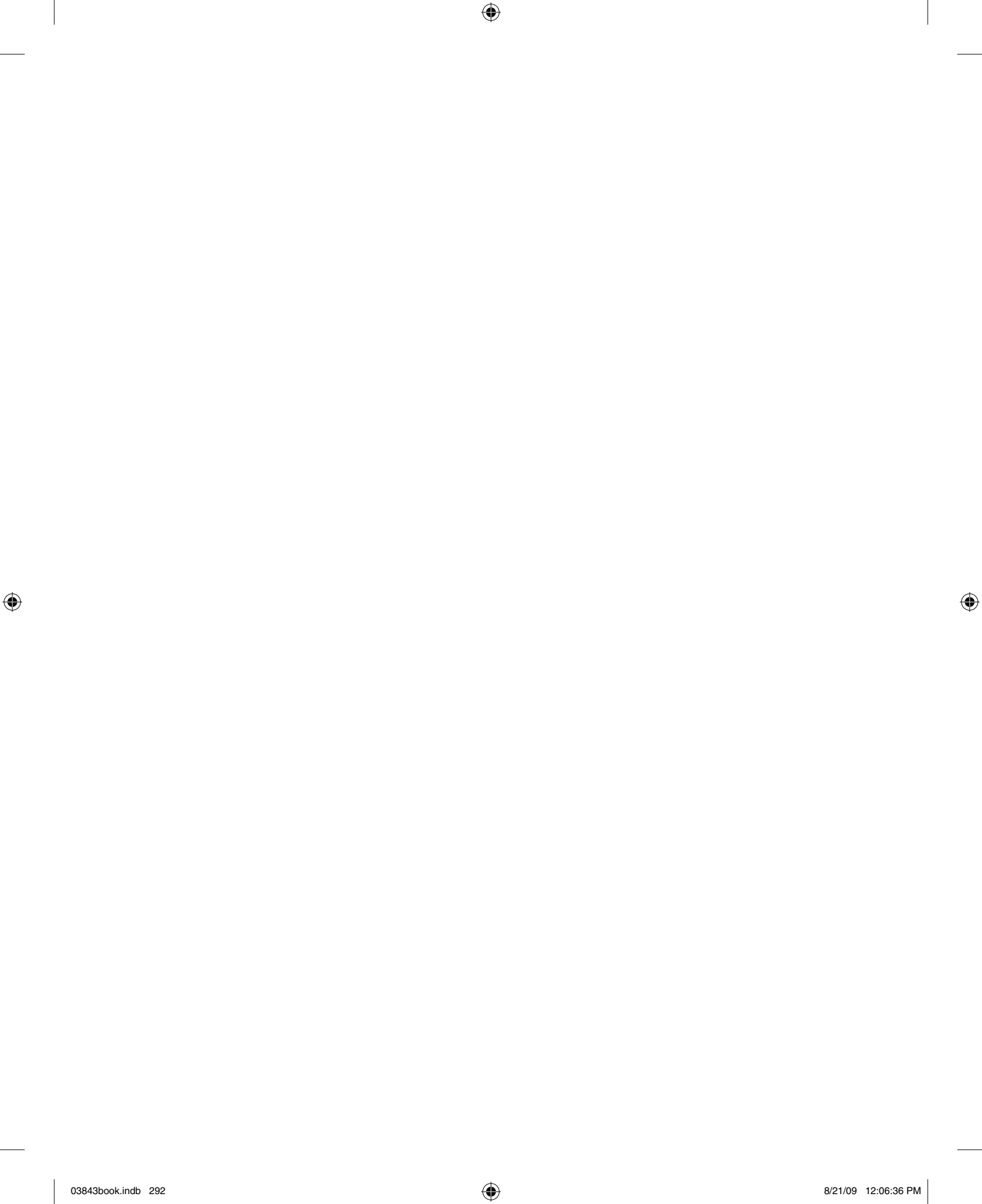
16. As an administrator, you want to increase the security on a Linux SMB/CIFS client system. You want to accomplish this by storing the authorization information in its own file, rather than in `/etc/fstab`. When this is done, what `/etc/fstab` mount option must you use to point to the file?
  - A. `certs=`
  - B. `securefile=`
  - C. `authorization=`
  - D. `credentials=`
17. A new server is arriving at the end of the week. It will have four 1TB hard drives installed and be configured in a RAID 5 array with no hot standby spare drives. How much data can be stored within this array?
  - A. 4TB
  - B. 3TB
  - C. 2TB
  - D. 1TB
18. You have been told by your manager that the server being moved from the test lab to production must have the two drives within it mirrored. What level of RAID is used for mirroring?
  - A. RAID 6
  - B. RAID 5
  - C. RAID 1
  - D. RAID 0
19. Which of the following commands would you type to summarize the quota information on all filesystems?
  - A. `repquota`
  - B. `repquota -a`
  - C. `quotacheck`
  - D. `quotacheck -a`
20. Which of the following tools may you use when creating partitions for Linux prior to installation? (Choose all that apply.)
  - A. Linux's `fdisk` from an emergency disk, run prior to the system installation
  - B. GNU Parted run from the system installation disk
  - C. A distribution-specific install-time utility
  - D. The DOS `FORMAT` utility, run prior to the system installation

## Answers to Review Questions

1. B. Logical partitions are numbered from 5 and up, and they reside inside an extended partition with a number between 1 and 4. Therefore, one of the first two partitions must be an extended partition that houses partitions 5 and 6. Because logical partitions are numbered starting at 5, their numbers won't change if `/dev/hda3` is subsequently added. The disk holds one primary, one extended, and two logical partitions.
2. A. A default use of `df` reports the percentage of disk space used. The number of inodes and filesystem types can both be obtained by passing parameters to `df`. This utility does *not* report how long a filesystem has been mounted.
3. A. The `/etc/fstab` file contains the mapping of partitions to mount points, so `/etc` must be an ordinary directory on the root partition, not on a separate partition. Options B and C describe restrictions that don't exist. Option D would be correct if `/etc` were not a separate partition.
4. D. Swap partitions aren't mounted in the way filesystems are, so they have no associated mount points.
5. C. Linux's `fdisk` doesn't write changes to disk until you exit from the program by typing `w`. Typing `q` exits without writing those changes, so typing `q` in this situation will avert disaster. Typing `w` would be precisely the wrong thing to do. Typing `u` would do nothing useful since it's not an undo command.
6. C. The `mkfs` command creates a new filesystem, overwriting any existing data and therefore making existing files inaccessible. This command does not set the partition type code in the partition table. The `-V` option is valid; it causes `mkfs` to be more verbose in reporting its activities. The `-t ext2` option tells `mkfs` to create an ext2 filesystem.
7. D. The `smbmount` program enables you to mount a remote SMB/CIFS share as if it were a local disk. Linux's NFS support would work if the Windows system were running an NFS server, but the question specifies that it's using SMB/CIFS, not NFS. An FTP server on the Windows system would enable file transfers but not direct file access. The same would be true for the Linux `smbclient` program.
8. A, D. A mount directory must be specified between the device entry (`/dev/hda8`) and the filesystem type code (`nfs`). The `nfs` filesystem type code may be used only with an NFS export specification of the form `server:export` as the device specification. Fields in `/etc/fstab` are separated by spaces or tabs, not commas (but commas are used between individual options if several options are specified in the options column). The `default` option may be used with *any* filesystem type.
9. B. A swap file may be located on local read/write filesystems. This includes, but is not limited to, the root filesystem. Swap space may *not* exist on NFS mounts (which are very slow compared to local disk partitions in any event). The amount of free disk space on the partition is irrelevant, as long as it's sufficient to support the swap file size.

10. C. It's easy to create a swap partition when adding a new disk, and in option C, the new user load might increase the need for memory and swap space, so adding a new swap partition is prudent. In options A and B, adding a swap partition would require downtime while juggling the partitions, and so it would disrupt use of the system. Adding a swap file makes more sense in those cases. In option D, adding swap space won't speed performance much (unless it's on a faster disk than the current swap space); a memory upgrade is in order to reduce reliance on swap space.
11. C. The `vgextend` command is used to add new physical volumes to an existing volume group, and option C shows the correct syntax for doing so with the specified volume group and physical volume. The `pvdisk` command displays information on physical volumes. The `vgcreate` command of option B would create a *new* volume group called `biglvm`, but that will fail with a volume group of that name already in existence. The `lvextend` command resizes a logical volume; you would use it as part of a resizing operation to add space to a logical volume. Option D's syntax for the use of this command is incorrect, and it's not the right command for the specified action, although it might be used not long thereafter to make the new disk space accessible.
12. A. QTParted is a GUI variant of the GNU Parted program. This program supports resizing several partition types, including FAT, ext2fs, ext3fs, and ReiserFS. (The GNOME Partition Editor, or GParted, is a similar tool but is not listed as an option.) GNU Parted is not a GUI tool, Part is a made-up program name, and `cfdisk` is a Linux text-mode partitioning tool that can't resize filesystems.
13. C. The `-t` option is used to tell `fsck` what filesystem to use. Normally, `fsck` determines the filesystem type automatically. The `-A` option causes `fsck` to check all the filesystems marked to be checked in `/etc/fstab`. The `-N` option tells `fsck` to take no action and to display what it would normally do, without actually doing it. The `-C` option displays a text-mode progress indicator of the check process.
14. D. The `free` utility would create the display shown. The `mt` command controls a tape device and does not produce output like this. The `df` utility is used to see the amount of free disk space, not memory use. The `swapon` utility enables swap space but does not produce a summary like this one.
15. C. The command given will cause the contents of `/dev/sdc5` to be mounted on `/home2` with the filesystem type autodetected and default options used.
16. D. Ordinarily, most SMB/CIFS shares require a username and password as a means of access control. The `credentials=file` mount option can be used to point Linux at a file that holds the username and sensitive password information.
17. B. In a RAID 5 array, the amount of data that can be stored is equal to the number of disks minus 1, since that amount of space will be used for holding parity information. (Hot standby spare drives further reduce available storage space, if used.) In this case, there are a total of four drives. Subtracting one means the amount of data space available is equal to three times the 1TB individual drive size, or a total of 3TB.

18. C. In a RAID 1 array, the disks are mirrored. RAID 5 and RAID 6 are both implementations of disk striping with parity, while RAID 0 is disk striping without parity.
19. B. The `repquota` utility is used to summarize the quota information on the filesystem. When used with the `-a` option, it will show this information for all filesystems. The `quotacheck` utility checks quota information on a disk and writes corrections.
20. A, B, C. You can usually define partitions using just about any tool that can create them, although with some tools (such as DOS's `FDISK`), you may need to change the partition type code using Linux tools. The `DOS FORMAT` utility is used to create a FAT filesystem, not to define a partition.





# Chapter 7

## Managing Packages and System Backups

---

### THE FOLLOWING COMPTIA OBJECTIVES ARE COVERED IN THIS CHAPTER:

- ✓ **1.8 Perform the following package management functions (Install, remove and update programs: rpm [rpm -Uvh, rpm -qa, rpm -e, yum], deb [dpkg -i, dpkg -r, apt-get, apt-cache search], source [ ./configure, make, make install, make uninstall, tar, make clean, autoconf, make test, tar.gz, INSTALL, bzip, gzip]; Resolve dependencies; Add and remove repositories).**
- ✓ **2.4 Conduct and manage backup and restore operations (Copying data: rsync and ftp; Archive and restore commands: cpio, tar, dump, restore, dd).**



Managing installed software involves a wide variety of tasks, many of which are specific to particular types of software or even individual packages. Other chapters cover some specific examples, such as network server configuration (Chapter 10, “Configuring Network Servers I” and Chapter 11, “Configuring Network Servers II”). This chapter covers the mechanics of package installation in general, using any of three common packaging schemes. This chapter also covers tools and procedures you can use to protect both system packages and user data by backing up the computer. Although they’re often overlooked, system backups are critical; having good backups can greatly reduce downtime in the event of a disk failure, a system break-in, or even just an accidental deletion of a critical file. Several backup options exist, in terms of both the hardware used to back up data and the software you use to do the job.

## Understanding Package Concepts

Any OS is defined largely by the files it installs on the computer. In the case of Linux, these files include the Linux kernel; critical utilities stored in directories like `/bin`, `/sbin`, `/usr/bin`, and `/usr/sbin`; and configuration files stored in `/etc`. How those files came to reside in their locations is irrelevant to the identity of the computer as a Linux box, but this detail is critically important to the day-to-day duties of a system administrator. When an updated version of a program is released, it’s extremely helpful to be able to track down the installed version of the program, determine just what version the installed program is, and update all the necessary files. A failure to do all of this can leave a system with two copies of a program or its support files, which can result in confusion. It’s also important that when you install a new program, you avoid accidentally overwriting files that belong to another program.

To help you keep track of installed programs, documentation, and so on, various package maintenance utilities have emerged. Some of these, such as the *RPM Package Manager (RPM)* and *Debian package tools*, are tightly woven into various Linux distributions, thus providing a centralized mechanism for program updates.

## File Collections

Most programs today consist of several files. Many programs come with one or more documentation files, configuration files, and support programs. For this reason, it’s long been common practice, on all platforms, to bundle related files together in one carrier file. This carrier file typically uses compression to save disk space and download time, and it may include information on the placement of specific files once they’re extracted and installed on the computer.

Linux package file formats all provide these useful features. A package file may contain a single program file or dozens (even hundreds or thousands) of files. A complete Linux distribution, in turn, consists of hundreds of package files, all designed to coexist and even work together to provide the features associated with Linux.

In addition to providing a common carrier mechanism for package transport, the RPM and Debian package systems provide a means of recording additional information about the package. This information includes a version number, a build number, the name of the package maintainer, the date and time of the package's last compilation, the hostname of the computer that built the package, one or more descriptions of the package, and a few other miscellaneous pieces of information. Typically, you can access all of this information either before or after installing a package on the computer, which can be quite helpful—you can read the package description to determine whether it's really what you want to install, before you do so.

## The Installed File Database

One of the problems with a simple file-collection mechanism is that there's no way to track what files you've installed, what files are associated with other files, and so on. It's easy for a system using such a simple package mechanism to fall into chaos or collect stray files. A partial solution to these problems is to maintain a centralized database of installed files, known as the *installed file database*, *package database*, or similar terms. Both the RPM and Debian systems provide this feature. With RPM, the database is stored in the `/var/lib/rpm` directory; for Debian packages, the database is in `/var/lib/dpkg`. Tarballs don't support a package database, although it's possible to have special programs track tarball installations, as Slackware does.



*Tarballs* are file collections created by the tar utility program. Although they lack some of the features of RPM and Debian packages, they're more universally compatible, and they're easier to create than RPM or Debian packages.

Most people don't need to understand the details of how the installed file database works; this information is most useful to those who write the tools or need to recover a seriously corrupted system. What is important are the features that the database provides to a Linux system, including the following:

**Package information** The supplementary information associated with a package—build date, description, version number, and so on—is copied from the package file to the installed file database when you install the package. This fact enables you to retrieve information even if you delete the original package file.

**File information** The database includes information on all the files installed on the computer via the package system. This information includes the name of the package to which the file belongs so that you can track a file back to its owner. There's also a checksum value and information on file ownership and permissions, which make it possible to detect when a file has been altered—assuming the database hasn't been tampered with. This file information does

*not* extend to any files users create or even to nonstandard configuration files for some packages. Standard configuration files are typically tracked, however.

**Dependencies** A *dependency* is a reliance of one package on another. For instance, many programs rely on `libc`. Packages include information on the files or packages on which they depend. This feature allows the package management system to detect these dependencies and prevent installation of a package if its dependencies are unmet. The system can also block the removal of a package if others depend on it.

**Provision information** Some packages provide features that are used by other packages. For instance, a mail client may rely on a mail server, and various mail servers exist for Linux. In this case, a simple file or package dependency can't be used because more than one mail server can be used to fulfill the client's requirements. Nonetheless, this feature is essentially a type of dependency.

Whenever you install, remove, or modify a package through a package management system, that system updates its database to reflect the changes you've made. You can then query the database about your installed packages, and the system can use the database when you subsequently modify your installation. In this way, the system can head off trouble—for instance, it can warn you and abort installation of a package if that package contains files that would overwrite files belonging to another package.

The package database does not include information on files or packages installed in any way but through the package management system. For this reason, it's best not to mix different types of packages. Although it's possible to install both RPM and Debian package management systems on one computer, their databases remain separate, thus reducing the benefits of conflict tracking, dependencies, and so on. For instance, you might install an important library in Debian format, but RPM packages that rely on that library won't know the library is installed, and so they will not install unless you provide an override switch. Further, you may not be warned that other programs require the library when you remove or upgrade it, so you might inadvertently break the RPM packages.

Some programs are distributed only in tarball form. In such cases, you can attempt to build an RPM or Debian package from the tarball or install from the tarball without the benefit of a package management system. Although the latter option has the drawbacks just outlined, it's often simpler than trying to create an RPM or Debian package. If you install only a few such programs, chances are you won't have too much trouble, especially if you keep good records on what you're installing from tarballs. Typically, programs you compile from source code go in the `/usr/local` directory tree, which isn't used by most RPM or Debian packages. This fact helps keep the two program types isolated, further reducing the chance of trouble.

## Using Network Repositories

Atop the local package database and package installation tools lies another layer. This layer is network-enabled, and it communicates with network repositories that hold a wide selection of software for your distribution. When you install software using such network-enabled tools, they check the network repository and, if necessary, download and install

dependencies. For instance, if you want to install the MegaWord program but it depends on the SuperSpell package, which you don't have installed, the network tools will automatically download and install SuperSpell.

You can also use network-enabled tools to check for updated software. Sometimes these updates are of minor importance; they fix minor bugs that might not even affect you. Other times, though, such updates fix security bugs. Thus, regularly using network-enabled tools to check for updates is good security practice.

## Rebuilding Packages

One of the features of package systems is that they enable you to either install a *binary package* (sometimes referred to as a *precompiled package*) or recompile a *source package* on your own system. The former approach is usually simpler and less time-consuming, but the latter approach enables you to customize a program. This customization can include both changes to the program source code and compile-time changes (such as compiling a package on an unusual architecture). Recompilation is possible both with the sophisticated RPM and Debian systems and with simpler tarballs—in fact, the primary means of source code distribution is usually as a tarball.

If you find a tarball for a package that is not available in other forms, you have two basic choices: you can compile or install the software as per the instructions in the tarball, which bypasses your RPM or Debian database if your distribution uses one, or you can create an RPM or Debian package from the original tarball and install the resulting binary package. The former approach is usually simpler when you want to install the package on just one system, despite the drawback of losing package database information. The latter approach is superior if you need to install the package on many similar systems, but it takes more effort—you must create special files to control the creation of a final RPM or Debian package and then use special commands to create that package.



The upcoming section “Compiling Source Code” covers the basics of compiling programs from source code. Creating binary RPMs and Debian packages from source code tarballs, though, is beyond the scope of this book. Consult the documentation for the package system for more information. In particular, the RPM HOWTO (<http://tldp.org/HOWTO/RPM-HOWTO>) contains this information for RPM. The book *Red Hat RPM Guide* by Eric Foster-Johnson (Wiley, 2003) may also be useful for those who need to delve deeply into the RPM system.

Source code is available in formats other than tarballs. Today, many program authors take the time to create *source RPMs*, which are source code packages meant to be processed by the RPM tools. Debian uses a control file, a patch file, and an original source code tarball as an equivalent to a source RPM. These files are most commonly found on sites catering specifically to Debian-based systems. A source RPM is easy to compile into a binary RPM for any given computer; all you need to do is call the `rpmbuild` program with the `--rebuild` argument and the name of the source package. (Sometimes additional arguments are needed,

such as when you are cross-compiling for one platform on another). This recompilation usually takes somewhere between a few seconds and several minutes, but it can take hours for large packages on slow computers. The result is one or more binary RPMs in the `/usr/src/redhat/RPMS/i386` directory or someplace similar (redhat may be something else on non-Red Hat distributions, and i386 is likely to be something else on non-x86 platforms or on distributions that optimize for Pentium or later CPUs).

However you do it, recompiling programs from source code has several advantages and disadvantages compared to using a ready-made binary package. One of the primary advantages is that you can control various compilation options, and you can even modify the source code to fix bugs or customize the program for your particular needs. Making such changes is much easier when you start with a tarball than when you start with an RPM or Debian source package, however. Another advantage is that you can compile a program for an unusual distribution. You might not be able to find a package of a particular program for Alpha or PowerPC architectures, for instance, but if a source package is available, you can compile it yourself. Similarly, if you compile a package yourself, you can work around some library incompatibilities you might encounter with prebuilt binaries, particularly if the binaries were created on a distribution other than the one you use.

The primary drawback to compiling your own packages is that it takes time. This problem is exacerbated if you need to install additional development libraries, compilers, or other tools in order to make a package compile. (Many programs need particular utilities to compile but not to run.) Sometimes a source package needs certain versions of other programs to compile, but you may have an incompatible version, making compilation impossible until you change the version you have. New Linux users also often have trouble with recompiling because of unfamiliarity with the procedures.



The Gentoo Linux distribution was designed to enable users to recompile the entire distribution relatively easily. This process takes many hours (sometimes longer than a day), though.

## Installing and Removing Packages

The three most common package formats in Linux are RPM packages, Debian packages, and tarballs. Of these three, tarballs are the most primitive, but they are also the most widely supported. Most distributions use either RPMs or Debian packages as the basis for most installed files. Therefore, it's important to understand how to use at least one of these two formats for most distributions, as well as tarballs. Compiling from source code has its own challenges.

### Handling RPM Packages

The most popular package manager in the Linux world is RPM. The RPM system provides all the basic tools described in the earlier section, “Understanding Package Concepts,” such as a package database that allows for checking conflicts and ownership of particular files.



## RPM Distributions and Conventions

RPM was developed by Red Hat for its own distribution, but it has since been adopted by others, such as Fedora, Mandriva, SUSE, and Yellow Dog. These distributions vary in many details other than their package management. This fact has consequences for package installation, since a package intended for one distribution might or might not install cleanly on another one.



Red Hat has splintered into two distributions: Fedora is the downloadable version favored by home users, students, and businesses on a tight budget. The *Red Hat* name is now reserved for the for-pay version of the distribution.

RPM is a cross-platform tool. Some non-Linux Unix systems can use RPM, although most don't use it as their primary package distribution system. RPM supports any CPU architecture, and RPM-based distributions are available for x86, x86-64, IA-64, PowerPC, and other CPUs. For the most part, source RPMs are transportable across architectures—you can use the same source RPM to build packages for any CPU you like. Some programs are actually composed of architecture-independent scripts, and so they need no recompilation. There are also documentation and configuration packages that work on any CPU.

The convention for naming RPM package files is as follows:

`packagename-a.b.c-x.arch.rpm`

Each of the filename components has a specific meaning:

**packagename** This is the name of the package, such as `samba` for the Samba file and print server.

**a.b.c** This is the package version number, such as 3.2.4. The version number doesn't have to be three period-separated numbers, but that's the most common form. The program author assigns the version number.

**x** The number following the version number is the *build number* (also known as the *release number*). This number represents minor changes made by the package maintainer, not by the program author. These changes may represent altered startup scripts or configuration files, changed file locations, added documentation, or patches appended to the original program to fix bugs or to make the program more compatible with the target Linux distribution. Some distribution maintainers add a letter code to the build number to distinguish their packages from those of others. Note that these numbers are *not* comparable across package maintainers—George's build number 5 of a package is *not* necessarily an improvement on Susan's build number 4 of the same package.

**arch** The final component preceding the `.rpm` extension is a code for the package's architecture. The `i386` architecture code represents a file compiled for any x86 CPU from the 80386 onward. Some packages include optimizations for Pentiums or above (`i586` or `i686`), x86-64 packages use the `x86_64` code, and binary packages for other CPUs use codes for their CPUs,



such as ppc for PowerPC CPUs. Scripts, documentation, and other CPU-independent packages generally use the noarch architecture code. The main exception to this rule is source RPMs, which use the src architecture code.



Most RPM-based distributions for x86-64 CPUs can use RPMs for both x86-64 and x86 CPUs. All other things being equal, you should favor the x86-64 RPMs, but if you can only find an x86 package (i386 or similar architecture codes), you can go ahead and install it. The software will run in 32-bit mode rather than 64-bit mode.

For instance, the Fedora 10 distribution ships with a Samba package called `samba-3.2.4-0.22.fc10.i386.rpm`, indicating that this is build 0.22.fc10 of Samba 3.2.4, compiled for any 80386 or above x86 CPU. These naming conventions are just that, though—conventions. It’s possible to rename a package however you like, and it will still install and work. The information in the filename is retained within the package.

In an ideal world, any RPM package will install and run on any RPM-based distribution that uses an appropriate CPU type. Unfortunately, compatibility issues can crop up from time to time. Different dependencies, naming conventions, startup script requirements, and other distribution-specific idiosyncrasies make using an RPM intended for one distribution a bit risky on another one. You might be able to get away with it, particularly for simple programs that don’t require SysV startup scripts; but whenever possible, you should try to find RPMs built for your particular distribution.

### Using *rpm* Commands

The main RPM utility program is known as `rpm`. Use this program to install or upgrade a package at the shell prompt. The `rpm` command has the following syntax:

`rpm [operation] [options] [package-files|package-names]`

Table 7.1 summarizes the most common `rpm` operations, and Table 7.2 summarizes the most important options. Be aware, however, that `rpm` is a very complex tool, so this listing is necessarily incomplete. Tables 7.1 and 7.2 do include information on the most common `rpm` features, however. For information on operations and options more obscure than those listed in Tables 7.1 and 7.2, see the `rpm` man pages. Many of `rpm`’s less-used features are devoted to the creation of RPM packages by software developers.

**TABLE 7.1** Common `rpm` Operations

| Operation | Description                                                                   |
|-----------|-------------------------------------------------------------------------------|
| -i        | Installs a package; system must <i>not</i> contain a package of the same name |
| -U        | Installs a new package or upgrades an existing one                            |



**TABLE 7.1** Common rpm Operations (*continued*)

| Operation            | Description                                                                                                                         |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------------|
| -F or --freshen      | Upgrades a package only if an earlier version already exists                                                                        |
| -q                   | Queries a package—finds if a package is installed, what files it contains, and so on                                                |
| -V or -y or --verify | Verifies a package—checks that its files are present and unchanged since installation                                               |
| -e                   | Uninstalls a package                                                                                                                |
| -b                   | Builds a binary package, given source code and configuration files; moved to the <code>rpmbuild</code> program with RPM version 4.2 |
| --rebuild            | Builds a binary package, given a source RPM file; moved to the <code>rpmbuild</code> program with RPM version 4.2                   |
| --rebuilddb          | Rebuilds the RPM database to fix errors                                                                                             |

**TABLE 7.2** Most Important rpm Options

| Option            | Used with Operations | Description                                                                                                                                                                                                            |
|-------------------|----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| --root <i>dir</i> | Any                  | Modifies the Linux system having a root directory located at <i>dir</i> . This option can be used to maintain one Linux installation discrete from another one (say, during OS installation or emergency maintenance). |
| --force           | -i, -U, -F           | Forces installation of a package even when it means overwriting existing files or packages.                                                                                                                            |
| -h or --hash      | -i, -U, -F           | Displays a series of hash marks (#) to indicate the progress of the operation.                                                                                                                                         |
| -v                | -i, -U, -F           | Used in conjunction with the -h option to produce a uniform number of hash marks for each package.                                                                                                                     |
| --nodeps          | -i, -U, -F, -e       | Performs no dependency checks. Installs or removes the package even if it relies on a package or file that's not present or is required by a package that's not being uninstalled.                                     |

**TABLE 7.2** Most Important rpm Options (*continued*)

| Option                               | Used with Operations | Description                                                                                     |
|--------------------------------------|----------------------|-------------------------------------------------------------------------------------------------|
| --test                               | -i, -U, -F           | Checks for dependencies, conflicts, and other problems without actually installing the package. |
| --prefix <i>path</i>                 | -i, -U, -F           | Sets the installation directory to <i>path</i> (works only for some packages).                  |
| -a or --all                          | -q, -V               | Queries or verifies all packages.                                                               |
| -f <i>file</i> or --file <i>file</i> | -q, -V               | Queries or verifies the package that owns <i>file</i> .                                         |
| -p <i>package-file</i>               | -q                   | Queries the uninstalled RPM <i>package-file</i> .                                               |
| -i                                   | -q                   | Displays package information, including the package maintainer, a short description, and so on. |
| -R or --requires                     | -q                   | Displays the packages and files on which this one depends.                                      |
| -l or --list                         | -q                   | Displays the files contained in the package.                                                    |

To use rpm, you combine one operation with one or more options. In most cases, you include one or more package names or package filenames as well. (A package filename is a complete filename, but a package name is a shortened version. For instance, a package filename might be `samba-3.2.4-0.22.fc10.i386.rpm`, while the matching package name is `samba`.) Either you can issue the rpm command once for each package or you can list multiple packages, separated by spaces, on the command line. The latter is often preferable when you're installing or removing several packages, some of which depend on others in the group. Issuing separate commands in this situation requires that you install the depended-on package first or remove it last, whereas issuing a single command allows you to list the packages on the command line in any order.

Some operations require that you give a package filename, and others require a package name. In particular, `-i`, `-U`, `-F`, and the rebuild operations require package filenames; `-q`, `-V`, and `-e` normally take a package name, although the `-p` option can modify a query (`-q`) operation to work on a package filename.

When installing or upgrading a package, the `-U` operation is generally the most useful because it enables you to install the package without manually uninstalling the old one. This one-step operation is particularly helpful when packages contain many dependencies because rpm detects these and can perform the operation should the new package fulfill the dependencies provided by the old one.



When upgrading your kernel, install the new one with the `-i` option rather than `-U`. This ensures that you'll still have the old kernel to boot, in case the new one gives you troubles.

To use `rpm` to install or upgrade a package, issue a command similar to the following:

```
rpm -Uvh samba-3.2.11-0.30.fc10.i386.rpm
```

You could also use `rpm -ivh` in place of `rpm -Uvh` if you don't already have a `samba` package installed.



It's possible to distribute the same program under different names. In this situation, upgrading may fail, or it may produce a duplicate installation, which can yield bizarre program-specific malfunctions. Red Hat has described a formal system for package naming to avoid such problems, but they still occur occasionally. Therefore, it's best to upgrade a package using a subsequent release provided by the same individual or organization that provided the original.

Verify that the package is installed with the `rpm -qi` command, which displays information such as when and on what computer the binary package was built. Listing 7.1 demonstrates this command. (`rpm -qi` also displays an extended plain-English summary of what the package is, which has been omitted from Listing 7.1.)

### Listing 7.1: RPM Query Output

```
$ rpm -qi samba
Name : samba Relocations: (not relocatable)
Version : 3.2.11 Vendor: Fedora Project
Release : 0.30.fc10 Build Date: Sat 18 Apr 2009 07:41:23 PM EDT
Install Date: Thu 23 Apr 2009 11:43:36 PM EDT Build Host: x86-7.fedora.phx.redhat.com
Group : System Environment/Daemons Source RPM: samba-3.2.11-0.30.fc10.src.rpm
Size : 11247377 License: GPLv3+ and LGPLv3+
Signature : DSA/SHA1, Mon 20 Apr 2009 02:05:00 PM EDT, Key ID bf226fcc4ebfc273
Packager : Fedora Project
URL : http://www.samba.org/
Summary : The Samba Suite of programs
```

## Using Yum

Yum (<http://linux.duke.edu/projects/yum/>) is one of several meta-packagers for RPM—Yum enables you to easily install a package and all its dependencies using a single

command line. When using Yum, you don't even need to locate and download the package files, because Yum does this for you by searching in one or more repositories (Internet sites that host RPM files for a particular distribution).

Yum originated with the fairly obscure Yellow Dog Linux distribution, but it's since been adopted by Red Hat, Fedora, and some other RPM-based distributions. Yum isn't used by all RPM-based distributions, though; SUSE and Mandriva, to name just two, each use their own meta-packagers. Debian-based distributions generally employ the Advanced Package Tools (APT), as described later in "Using APT." Nonetheless, because of the popularity of Red Hat and Fedora, knowing Yum can be valuable.

The most basic way to use Yum is with the `yum` command, which has the following syntax:

```
yum [options] [command] [package...]
```

Which options are available depend on the command you use. Table 7.3 describes common `yum` commands.

**TABLE 7.3** Common `yum` Commands

| Command                                            | Description                                                                                                                                                                                                                                                  |
|----------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>install</code>                               | Installs one or more packages by package name. Also installs dependencies of the specified package or packages.                                                                                                                                              |
| <code>update</code>                                | Updates the specified package or packages to the latest available version. If no packages are specified, <code>yum</code> updates every installed package.                                                                                                   |
| <code>check-update</code>                          | Checks to see whether updates are available. If they are, <code>yum</code> displays their names, versions, and repository area (updates or extras, for instance).                                                                                            |
| <code>upgrade</code>                               | Works like <code>update</code> with the <code>--obsoletes</code> flag set, which handles obsolete packages in a way that's superior when performing a distribution version upgrade.                                                                          |
| <code>remove</code> or <code>erase</code>          | Deletes a package from the system; similar to <code>rpm -e</code> , but <code>yum</code> also removes depended-on packages.                                                                                                                                  |
| <code>list</code>                                  | Displays information about a package, such as the installed version and whether an update is available.                                                                                                                                                      |
| <code>provides</code> or <code>whatprovides</code> | Displays information about packages that provide a specified program or feature. For instance, typing <code>yum provides samba</code> lists all the Samba-related packages, including every available update. Note that the output can be copious.           |
| <code>search</code>                                | Searches package names, summaries, packagers, and descriptions for a specified keyword. This is useful if you don't know a package's name but can think of a word that's likely to appear in one of these fields but not in these fields for other packages. |

**TABLE 7.3** Common yum Commands (*continued*)

| Command      | Description                                                                                                                                                                                                  |
|--------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| info         | Displays information about a package, similar to the <code>rpm -qi</code> command.                                                                                                                           |
| clean        | Cleans up the Yum cache directory. Running this command from time to time is advisable, lest downloaded packages chew up too much disk space.                                                                |
| shell        | Enters the Yum shell mode, in which you can enter multiple Yum commands one after another.                                                                                                                   |
| resolvedep   | Displays packages matching the specified dependency.                                                                                                                                                         |
| localinstall | Installs the specified local RPM files, using your Yum repositories to resolve dependencies.                                                                                                                 |
| localupdate  | Updates the system using the specified local RPM files, using your Yum repositories to resolve dependencies. Packages other than those updated by local files and their dependencies are <i>not</i> updated. |
| deplist      | Displays dependencies of the specified package.                                                                                                                                                              |

In most cases, using Yum is easier than using RPM directly to manage packages, because Yum finds the latest available package, downloads it, and installs any required dependencies. Yum has its limits, though; it's only as good as its repositories, so it can't install software that's not stored in those repositories.

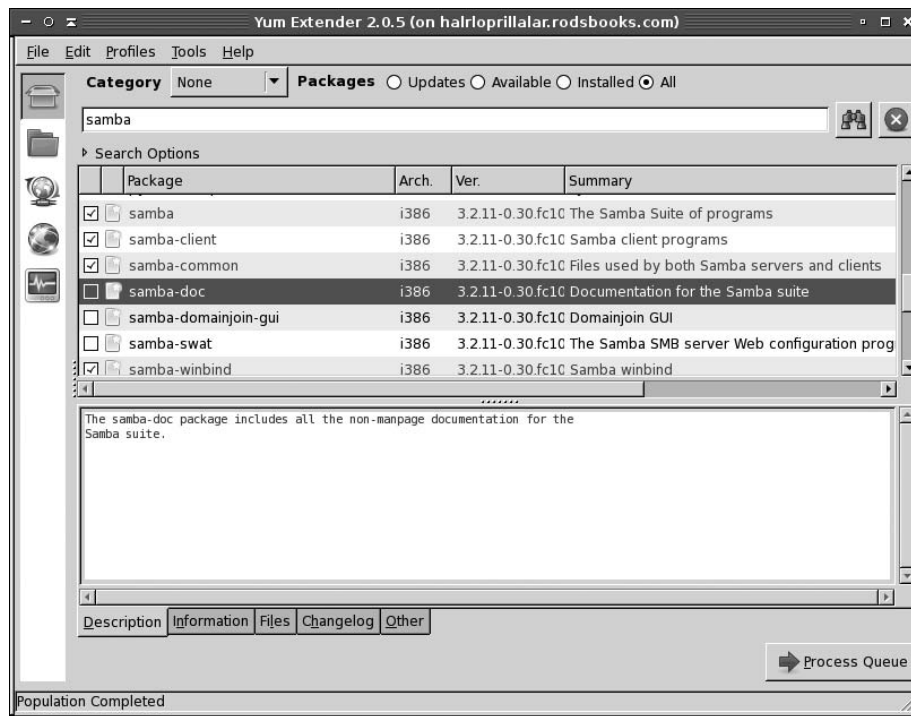


If you use Yum to automatically upgrade all packages on your system, you're effectively giving control of your system to the distribution maintainer. Although Red Hat or other distribution maintainers are unlikely to try to break into your computer in this way, an automatic update with minimal supervision on your part could easily break something on your system, particularly if you've obtained packages from unusual sources in the past.

If you don't want to install the package but merely want to obtain it, you can use `yumdownloader`. Type this command followed by the name of a package, and the latest version of the package will be downloaded to the current directory. This can be handy if you need to update a system that's not connected to the Internet; you can use another system that runs the same distribution to obtain the packages and then transfer them to the target system.

If you prefer to use GUI tools rather than command-line tools, you should be aware that GUI front-ends to yum exist. Examples include Yum Extender (yumex) and kyum. You can use the text-mode yum to install these front-ends, as in **yum install yumex**. Figure 7.1 shows Yum Extender in action. To use it, click the Updates, Available, Installed, or All button to show the relevant types of packages. You can enter a search term in the line below these options (Figure 7.1 shows *samba* in this field) to restrict the package selection. You can then check or uncheck the tick boxes next to specific package names to mark them for installation or removal. When you've made your selections, click Process Queue to implement the changes.

**FIGURE 7.1** The Yum Extender (yumex) program provides a GUI front-end to Yum.



Yum is configured via the `/etc/yum.conf` file, with additional configuration files in the `/etc/yum.repos.d/` directory. The `yum.conf` file holds basic options, such as the directory to which Yum downloads RPMs and where Yum logs its activities. Chances are you won't need to modify this file. The `/etc/yum.repos.d/` directory, on the other hand, potentially holds several files, each of which describes a Yum repository—that is, a site that holds RPMs that may be installed via Yum. You probably shouldn't directly edit these files; instead, if you want to add a repository, manually download the RPM that includes the repository configuration and install it using `rpm`. The next time you use Yum, it will access

your new repository along with the old ones. Several Yum repositories exist, mostly for Red Hat and Fedora, such as the following:

**Livna** This repository (<http://rpm.livna.org/r1owiki/>) hosts multimedia tools, such as additional codecs and video drivers.

**KDE Red Hat** Red Hat and Fedora favor the GNU Network Object Model Environment (GNOME) desktop environment, although they ship with the K Desktop Environment (KDE). The repository at <http://kde-redhat.sourceforge.net> provides improved KDE RPMs for those who favor KDE.

**Fresh RPMs** This repository (<http://freshrpms.net>) provides additional RPMs, mostly focusing on multimedia applications and drivers.

Many additional repositories exist. Try a Web search on terms such as *yum repository*, or check any Web site that hosts unusual software you want to run to see whether it provides a Yum repository. If so, it should provide an RPM or other instructions on adding its site to your Yum repository list.



The RPMFind Web site, <http://rpmfind.net>, is an extremely useful resource when you want to find an RPM of a specific program, but it's not accessible as a Yum repository. RPMFind includes links to RPMs built by programs' authors, specific distributions' RPMs, and those built by third parties.

## Handling Debian Packages

In their overall features, Debian packages are similar to RPMs, but the details of operation for each differ, and Debian packages are used on different distributions than are RPMs. Because each system uses its own database format, RPMs and Debian packages aren't interchangeable without converting formats.

### Debian Package Conventions

As the name implies, Debian packages originated with the Debian distribution. Since that time, the format has been adopted by several other distributions, including Ubuntu and Xandros. Such distributions are derived from the original Debian, which means that packages from the original Debian are likely to work well on other Debian-based systems. Although Debian doesn't emphasize flashy GUI installation or configuration tools, its derivatives add GUI configuration tools to the base Debian system, which makes these distributions more appealing to Linux novices. The original Debian favors a system that's as bug-free as possible, and it tries to adhere strictly to open source software principles rather than invest effort in GUI configuration tools.

Like RPM, the Debian package format is neutral with respect to both OS and CPU type. Debian packages are extremely rare outside Linux, though.

The original Debian distribution has been ported to many different CPUs, including x86, x86-64, IA-64, PowerPC, Alpha, 680x0, MIPS, and SPARC. The original architecture was x86, and subsequent ports exist at varying levels of maturity. Derivative distributions generally work only on x86 and x86-64 systems, but this could change in the future.

Debian packages follow a naming convention similar to those for RPMs, but Debian packages sometimes omit codes in the filename to specify a package’s architecture, particularly on x86 packages. When these codes are present, they may differ from RPM conventions. For instance, a filename ending in `i386.deb` indicates an x86 binary, `amd64.deb` is an x86-64 binary, and `all.deb` indicates a CPU-independent package, such as documentation or scripts. As with RPM files, this file-naming convention is only that—a convention. You can rename a file as you see fit. There is no code for Debian source packages because Debian source packages actually consist of several separate files.



To install an x86 Debian package on an x86-64 (aka AMD64) system, you must use the `--force-architecture` option to `dpkg`. This contrasts with RPM-based distributions, which usually permit such installation by default.

### Using *dpkg* Commands

Debian packages are incompatible with RPM packages, but the basic principles of operation are the same across both package types. Like RPMs, Debian packages include dependency information, and the Debian package utilities maintain a database of installed packages, files, and so on. You use the `dpkg` command to install a Debian package. This command’s syntax is similar to that of `rpm`:

`dpkg [options][action] [package-files|package-name]`

*action* is the action to be taken; Table 7.4 summarizes common actions. The options (Table 7.5) modify the behavior of the action, much like the options to `rpm`.

**TABLE 7.4** `dpkg` Primary Actions

| Action                                        | Description                                                                                      |
|-----------------------------------------------|--------------------------------------------------------------------------------------------------|
| <code>-i</code> or <code>--install</code>     | Installs a package                                                                               |
| <code>--configure</code>                      | Reconfigures an installed package—runs the post-installation script to set site-specific options |
| <code>-r</code> or <code>--remove</code>      | Removes a package but leaves configuration files intact                                          |
| <code>-P</code> or <code>--purge</code>       | Removes a package, including configuration files                                                 |
| <code>-p</code> or <code>--print-avail</code> | Displays information about an installed package                                                  |



**TABLE 7.4** dpkg Primary Actions (*continued*)

| Action                                       | Description                                                                 |
|----------------------------------------------|-----------------------------------------------------------------------------|
| -I or --info                                 | Displays information about an uninstalled package file                      |
| -l <i>pattern</i> or --list <i>pattern</i>   | Lists all installed packages whose names match <i>pattern</i>               |
| -L or --listfiles                            | Lists the installed files associated with a package                         |
| -S <i>pattern</i> or --search <i>pattern</i> | Locates the package(s) that own the file(s) specified by <i>pattern</i>     |
| -C or --audit                                | Searches for partially installed packages and suggests what to do with them |

**TABLE 7.5** Options for Fine-Tuning dpkg Actions

| Option                           | Used with Actions | Description                                                                                                                                                                                             |
|----------------------------------|-------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| --root= <i>dir</i>               | All               | Modifies the Linux system using a root directory located at <i>dir</i> . Can be used to maintain one Linux installation discrete from another one, say during OS installation or emergency maintenance. |
| -B or --auto-deconfigure         | -r                | Disables packages that rely on one that is being removed.                                                                                                                                               |
| --force- <i>things</i>           | Assorted          | Forces specific actions to be taken. Consult the dpkg man page for details of <i>things</i> this option does.                                                                                           |
| --ignore-depends= <i>package</i> | -i, -r            | Ignores dependency information for the specified package.                                                                                                                                               |
| --no-act                         | -i, -r            | Checks for dependencies, conflicts, and other problems without actually installing or removing the package.                                                                                             |
| --recursive                      | -i                | Installs all packages that match the package name wildcard in the specified directory and all subdirectories.                                                                                           |

**TABLE 7.5** Options for Fine-Tuning dpkg Actions *(continued)*

| Option                    | Used with Actions | Description                                                                              |
|---------------------------|-------------------|------------------------------------------------------------------------------------------|
| -G                        | -i                | Doesn't install the package if a newer version of the same package is already installed. |
| -E or --skip-same-version | -i                | Doesn't install the package if the same version of the package is already installed.     |

As with rpm, dpkg expects a package name in some cases and a package filename in others. Specifically, `--install (-i)` and `--info (-I)` both require the package filename, but the other commands take the shorter package name.

As an example, consider the following command, which installs the `samba-common_3.0.28a-1ubuntu4.7_amd64.deb` package:

```
dpkg -i samba-common_3.0.28a-1ubuntu4.7_amd64.deb
```

If you're upgrading a package, you may need to remove an old package before installing the new one. To do this, use the `-r` option to dpkg:

```
dpkg -r samba
```

To find information on an installed package, use the `-p` parameter to dpkg, as shown in Listing 7.2. This listing omits an extended English description of what the package does.

**Listing 7.2:** *dpkg* Package Information Query Output

```
$ dpkg -p samba-common
Package: samba-common
Priority: optional
Section: net
Installed-Size: 7112
Maintainer: Ubuntu Core Developers <ubuntu-devel-discuss@lists.ubuntu.com>
Architecture: amd64
Source: samba
Version: 3.0.28a-1ubuntu4.7
Replaces: samba (<< 3.0.20b-1)
Depends: debconf (>= 0.5) | debconf-2.0, libc6 (>= 2.4), libcomerr2
(>= 1.33-3), libkrb53 (>= 1.6.dfsg.2), libldap-2.4-2 (>= 2.4.7), libncurses5
(>= 5.6+20071006-3), libpam-modules, libpopt0 (>= 1.10), libreadline5
(>= 5.2), libuuid1, ucf
Size: 3058414
```

Debian-based systems often use a somewhat higher-level utility called `dselect` to handle package installation and removal. The `dselect` utility provides a text-mode list of installed packages and packages available from a specified source (such as a CD-ROM drive or an FTP site), and it allows you to select which packages you want to install and remove. This interface can be very useful when you want to install several packages, but `dpkg` is often more convenient when manipulating just one or two packages. Because `dpkg` can take package filenames as input, it's also the preferred method of installing a package that you download from an unusual source or create yourself.

## Using APT

Advanced Package Tool utilities are another option for Debian package management. These tools center around the `apt-get` utility, which enables you to perform easy upgrades of packages, especially if you have a fast Internet connection. Debian-based systems include a file, `/etc/apt/sources.list`, that specifies locations from which important packages can be obtained. If you installed the OS from a CD-ROM drive, this file will initially list directories on the installation CD-ROM in which packages can be found. There are also likely to be a few lines near the top, commented out with hash marks (`#`), indicating directories on an FTP or Web site from which you can obtain updated packages. (These lines may be uncommented if you did a network install initially.)



Although APT is most strongly associated with Debian systems, a port to RPM-based systems is also available. Check <http://apt4rpm.sourceforge.net> for information on this port.

The `apt-get` utility works by obtaining information on available packages from the sources listed in `/etc/apt/sources.list` and then using that information to upgrade or install packages. The syntax is similar to that of `dpkg`:

```
apt-get [options][command] [package-names]
```

Table 7.6 describes the `apt-get` commands, and Table 7.7 describes the most commonly used options. In most cases, you won't actually use *any* options with `apt-get`, just a single command and possibly one or more package names. One particularly common use of this utility is to keep your system up-to-date with any new packages. The following two commands will accomplish this goal, if `/etc/apt/sources.list` includes pointers to up-to-date file archive FTP sites:

```
apt-get update
apt-get dist-upgrade
```

**TABLE 7.6** apt-get Commands

| Command         | Description                                                                                                                                                                                                                    |
|-----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| update          | Obtains updated information on packages available from the installation sources listed in <code>/etc/apt/sources.list</code> .                                                                                                 |
| upgrade         | Upgrades all installed packages to the newest versions available, based on locally stored information on available packages.                                                                                                   |
| dselect-upgrade | Performs any changes in package status (installation, removal, etc.) left undone after running <code>dselect</code> .                                                                                                          |
| dist-upgrade    | Similar to <code>upgrade</code> but performs “smart” conflict resolution to avoid upgrading a package if that would break a dependency.                                                                                        |
| install         | Installs a package by the package name (not by package filename), obtaining the package from the source that contains the most up-to-date version.                                                                             |
| remove          | Removes a specified package by the package name.                                                                                                                                                                               |
| source          | Retrieves the newest available source package file by the package filename, using information on available packages and installation archives listed in <code>/etc/apt/sources.list</code> .                                   |
| check           | Checks the package database for consistency and broken package installations.                                                                                                                                                  |
| clean           | Performs housekeeping to help clear out information on retrieved files from the Debian package database. If you don’t use <code>dselect</code> for package management, run this from time to time in order to save disk space. |
| autoclean       | Similar to <code>clean</code> but removes information only on packages that can no longer be downloaded.                                                                                                                       |

**TABLE 7.7** Most Useful apt-get Options

| Option                | Used with Commands                        | Description                                                     |
|-----------------------|-------------------------------------------|-----------------------------------------------------------------|
| -d or --download-only | upgrade, dselect-upgrade, install, source | Downloads package files but does not install them.              |
| -f or --fix-broken    | install, remove                           | Attempts to fix a system on which dependencies are unsatisfied. |

**TABLE 7.7** Most Useful apt-get Options (*continued*)

| Option                                                        | Used with Commands                                | Description                                                                                                               |
|---------------------------------------------------------------|---------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------|
| -m, --ignore-missing, or --fix-missing                        | upgrade, dselect-upgrade, install, remove, source | Ignores all package files that can't be retrieved (because of network errors, missing files, or the like).                |
| -q or --quiet                                                 | All                                               | Omits some progress indicator information. May be doubled (for instance, -qq) to produce still less progress information. |
| -s, --simulate, --just-print, --dry-run, --recon, or --no-act | All                                               | Performs a simulation of the action without actually modifying, installing, or removing files.                            |
| -y, --yes, or --assume-yes                                    | All                                               | Produces a "yes" response to any yes/no prompt in installation scripts.                                                   |
| -b, --compile, or --build                                     | source                                            | Compiles a source package after retrieving it.                                                                            |
| --no-upgrade                                                  | install                                           | Causes apt-get to <i>not</i> upgrade a package if an older version is already installed.                                  |

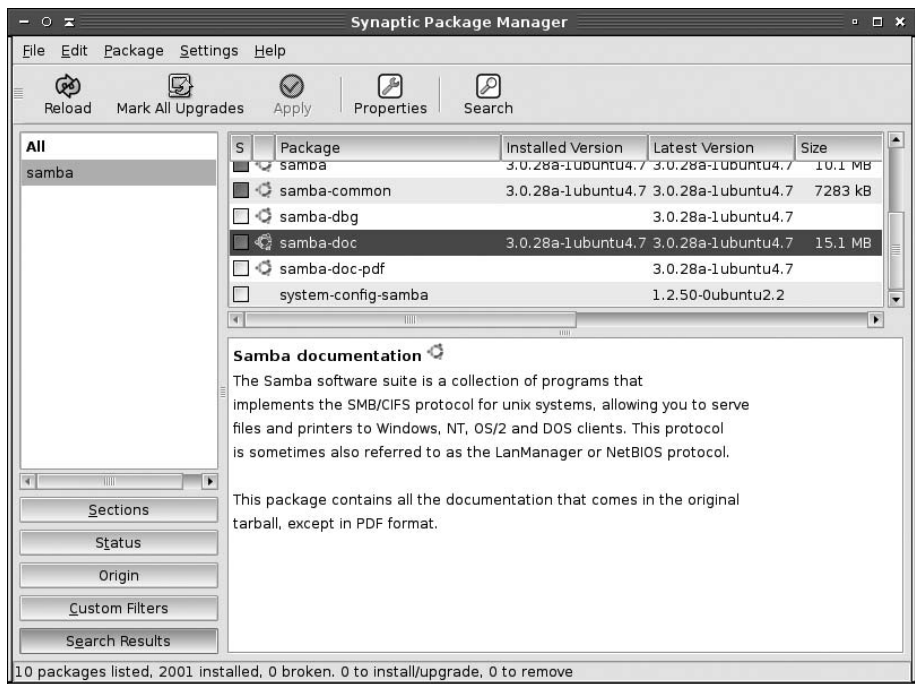


If you use apt-get to automatically upgrade all packages on your system, you are effectively giving control of your system to the distribution maintainer. Although Debian or other distribution maintainers are unlikely to try to break into your computer in this way, an automatic update with minimal supervision on your part could easily break something on your system, particularly if you've obtained packages from unusual sources in the past.

A GUI front-end to APT, Synaptic, is available. It presents a list of available packages; or you can use the Search button to narrow the range. When you've found a package you want to install or remove, click the box to the left of its name. When you've made all your selections, click Apply to have Synaptic make the changes. Figure 7.2 shows Synaptic in use.

Several additional tools are part of the APT suite, including apt-key, apt-sortpkgs, and apt-cache. Most of these tools are highly technical; however, apt-cache has a useful search function, activated by the search keyword, which searches the APT database's names and descriptions. For instance, typing **apt-cache search samba** finds all packages that explicitly mention Samba.

**FIGURE 7.2** Synaptic provides a GUI front-end to the APT utilities.



## Handling Tarballs

All distributions can use tarballs—files collected together with the `tar` utility and typically compressed with `compress`, `gzip`, or `bzip2`. Like RPM and Debian packages, tarballs may contain source code, binary files, or architecture-independent files such as documentation or fonts. These files lack dependency information, however, and `tar` maintains no database of installed files. Therefore, it's harder to remove programs installed via tarballs than it is to remove RPM or Debian packages. Slackware, though, maintains a database of files installed via Slackware's tarballs and the Slackware `pkgtool` utility.

## The Role of *tar* and Tarballs

A multipurpose tool, `tar` was originally created for archiving files to tape—the name stands for “tape archiver.” (The upcoming section “Backing Up and Restoring a Computer” describes this role of `tar` in more detail.) Because Unix, and hence Linux, treats hardware devices as files, a tape-archiving program like `tar` can be used to create archives as files on disk. These files can then be compressed, copied to floppy disk or other removable media, sent over a network, and so on.

In the Linux world, tarballs fill a role that's similar to that of zip files in the Windows world. There are differences, however. Zip utilities (including the `zip` and `unzip` commands in Linux) compress files and then add them to the archive. By contrast, `tar` does not directly support compression, so to compress files, the resulting archive is compressed with a second utility, such as `gzip`, `bzip2`, or `compress`. The `gzip` and `bzip2` programs are the most popular on Linux systems, although `compress` is still used on some older Unix systems. (The `gzip` utility can uncompress old `compress` archives.) The resulting file may have two extensions (such as `.tar.gz` or `.tar.bz2`), or that dual extension may be combined into a single, three-character extension (`.tgz` or `.tbz`) for easy storage on filesystems (like DOS's FAT) that don't support longer or multiple extensions. The older `compress` archives used an uppercase `Z` extension, so these tarballs have `.tar.Z` extensions.

Considered as a package distribution mechanism, tarballs are used primarily by the Slackware distribution, which is the oldest of the major Linux distributions still in common use. Slackware eschews flashy configuration tools in favor of a bare-bones approach. In this respect, Slackware resembles Debian, but Slackware uses custom extensions to `tar` as a way of tracking package installations. As noted earlier, Debian also uses source tarballs as part of its source package management system, but most administrators don't need to be concerned with this detail.

Although most other distributions don't rely on tarballs, they can be used with any distribution. Tarballs are particularly likely to be useful when you're faced with the task of compiling a program from source code, and especially if you must modify that source code for your system. The upcoming section "Compiling Source Code" describes how to compile software delivered as source code in a tarball.

Binary tarballs contain precompiled programs. Sometimes the tarball contains the program files in a form that enables you to expand the tarball directly into a target directory. For instance, you could change to the `/usr/local` directory and uncompress the tarball to have the program files dropped directly into `/usr/local/bin`, `/usr/local/man`, and so on. Other times you may need to uncompress the tarball in a temporary directory and then run an installation utility to install the software.



If you're unsure of how to proceed with a tarball installation, extract it into a temporary directory, and look for instructions. Sometimes you'll find separate installation instructions on the program's Web site or on the FTP site from which you obtained the software.

## Using *tar* Commands

The `tar` program is a complex package with many options. Most of what you'll do with the utility, however, can be covered with a few common commands. Table 7.8 describes the primary `tar` commands, and Table 7.9 describes the qualifiers for these commands that modify what the command does. Whenever you run `tar`, you use exactly one command, and you usually use at least one qualifier.

**TABLE 7.8** tar Commands

| Command             | Abbreviation | Description                                           |
|---------------------|--------------|-------------------------------------------------------|
| --create            | c            | Creates an archive                                    |
| --concatenate       | A            | Appends tar files to an archive                       |
| --append            | r            | Appends non-tar files to an archive                   |
| --update            | u            | Appends files that are newer than those in an archive |
| --diff or --compare | d            | Compares an archive to files on disk                  |
| --list              | t            | Lists archive contents                                |
| --extract or --get  | x            | Extracts files from an archive                        |

**TABLE 7.9** tar Qualifiers

| Command                             | Abbreviation               | Description                                                                                      |
|-------------------------------------|----------------------------|--------------------------------------------------------------------------------------------------|
| --directory <i>dir</i>              | C                          | Changes to directory <i>dir</i> before performing operations                                     |
| --file [ <i>host:</i> ] <i>file</i> | f                          | Uses file called <i>file</i> on computer called <i>host</i> as the archive file                  |
| --listed-incremental <i>file</i>    | g                          | Performs incremental backup or restore, using <i>file</i> as a list of previously archived files |
| --one-file-system                   | l (on older versions only) | Backs up or restores only one filesystem (partition)                                             |
| --multi-volume                      | M                          | Creates or extracts a multitape archive                                                          |
| --tape-length <i>N</i>              | L                          | Changes tapes after <i>N</i> kilobytes                                                           |
| --same-permissions                  | p                          | Preserves all protection information                                                             |
| --absolute-paths                    | P                          | Retains the leading / on filenames                                                               |



**TABLE 7.9** tar Qualifiers (*continued*)

| Command                    | Abbreviation                        | Description                                                                                               |
|----------------------------|-------------------------------------|-----------------------------------------------------------------------------------------------------------|
| --verbose                  | v                                   | Lists all files read or extracted; when used with --list, displays file sizes, ownership, and time stamps |
| --verify                   | W                                   | Verifies the archive after writing it                                                                     |
| --exclude <i>file</i>      | (none)                              | Excludes <i>file</i> from the archive                                                                     |
| --exclude-from <i>file</i> | X                                   | Excludes files listed in <i>file</i> from the archive                                                     |
| --gzip or --ungzip         | z                                   | Processes archive through gzip                                                                            |
| --bzip2                    | j (some older versions used I or y) | Processes archive through bzip2                                                                           |

Of the commands listed in Table 7.8, the most commonly used are --create, --extract, and --list. The most useful qualifiers from Table 7.9 are --file, --listed-incremental, --one-file-system, --same-permissions, --gzip, --bzip2, and --verbose. If you fail to specify a filename with the --file qualifier, tar will attempt to use a default device, which is often (but not always) a tape device file.

A typical tar command to extract files from a tarball looks like this:

```
tar --extract --verbose --gunzip --file samba-3.2.11.tar.gz
```

This command can be expressed somewhat more succinctly using command abbreviations:

```
tar xvfz samba-3.2.11.tar.gz
```

In either form, this tar command extracts files from `samba-3.2.11.tar.gz` to the current directory. Most tarballs include entire directory trees, so this command results in one or more directories being created, if they don't already exist, as well as files within the directories.



Before extracting a tarball, use the --list command to find out what files and directories it contains. This information can help you locate the files stored in the tarball. In addition, it can help you spot problems before they would occur in case a tarball does *not* contain a neat directory structure but instead contains files that would all be dropped in the current directory.

## Creating Tarballs

One feature of `tar` that you may find valuable is that the program can be used to easily create packages, as well as extract files from them. You can use this feature to move data files, documentation, or programs you've written or built yourself. Of course, you can also create RPM or Debian packages, but this process is more complex, and the usual method of doing this requires that you provide a tarball of the source code to begin with. It's easiest to create a tarball of all the files in a single directory:

```
tar cvzf my-stuff.tgz my-stuff-dir
```

You can then move the tarball to a removable medium, upload it to an Internet site, or just leave it on your hard disk. When you extract the files from the tarball, `tar` will create the original target directory and restore all the files and subdirectories from within it.

## Compiling Source Code

Linux's open source nature means that source code is available for most or all of the programs you run. This fact is a curiosity to some users, but it's extremely valuable to others. For instance, you might be able to fix a minor bug or change a default value by modifying the source code and recompiling it. Other times, you may be forced to compile a program from source code—say, if it's an obscure program that's not available in binary form for your computer's CPU.

## Procedures for Compiling and Installing Source Code

Source code can be compiled either as part of a source package for your package format (such as a source RPM) or from an original tarball provided by the program author. In either case, you should see to several prerequisites before compiling a program:

**Appropriate compilers** Source code requires one or more compilers and related programs. Most commonly, the GNU Compiler Collection (GCC) is needed, but sometimes other tools are needed. The package's documentation should detail these requirements. GCC is installed by default on many Linux systems, but if it's not installed on yours, use `Yum`, `apt-get`, or similar tools to install it. Some programs are written in interpreted, rather than compiled, languages. These require you to have an appropriate interpreter package, such as Perl or Python, rather than a compiler.

**make** Most compiled programs use a utility called `make` to direct the compilation process. As with GCC, `make` is usually installed by default on Linux systems, but you may want to double-check this detail.

**Support libraries and header files** All Linux programs rely on one or more *libraries*, which provide support functions used by many different programs. Before you can compile a program, your system must have all the libraries upon which it relies. What's more, compiling a program requires that you have a set of *header files*, which are files that tell a program how to call library functions. On most distributions, the header files are installed separately from the

libraries themselves. Typically, the header file package name is similar to that for the library but includes a term such as `dev` or `devel` in the package name. The documentation for the program you're compiling should detail the libraries it requires, but if you're installing from a source tarball, the documentation may not tell you the precise package name you'd install for your distribution; you may need to hunt a bit and perhaps guess.

**System resources** Compiling a program takes disk space, CPU time, and RAM. For most programs, the resources required aren't huge by modern standards, but you may want to check, particularly when compiling large programs. Disk space can become an issue with big packages, and the CPU time consumed might interfere with other uses of the system. For this reason, you may want to compile programs at off times, if the computer is normally used for other tasks.

When compiling a source RPM, you pass the `--rebuild` option to `rpmbuild`, along with the source RPM name. With luck, the process will complete without errors. If it doesn't, you face the daunting task of troubleshooting the problem. Most frequently, the issue is a missing development library package (that is, library headers) or development tool. Scrolling back over the output of the build process should yield a clue, such as a comment that a library wasn't present on your system.

Compiling a package from a source tarball cannot be easily summarized in a simple procedure, because the procedure varies from one package to another. After uncompressing the package, you should search for a file called `README`, `INSTALL`, `CONFIGURE`, or something similar. This file should describe the configuration and installation process. Frequently, the source package includes a script called `configure`, which is generated by the programmer using a utility called `autoconf`. You type `./configure` to have the package autodetect your computer's installed libraries and configure itself appropriately. You may be able to pass options to this script to further customize it—say, to add or remove support for a particular protocol or feature. Read the documentation to learn about such options, because they're highly package-specific.



Some packages are poorly documented. In such cases, reading the `configure` script in a text editor may give you some idea about the options it accepts.

Some programs have no `configure` script but provide similar functionality through some other means. The Linux kernel itself is one such program—you type **`make config`**, **`make menuconfig`**, or **`make xconfig`** to configure it using text-mode, menu-based text-mode, or GUI tools, respectively. This process is tedious. The upcoming section “Special Procedures for the Kernel and Drivers” touches on this topic.

Some programs (particularly small ones) don't use configuration scripts. To change their options, you must typically edit a file called `Makefile` or `makefile`. Precisely what you might want to change in this file is highly package-specific, so consult its documentation for details.

You can compile most programs by typing **`make`** in their source directories. This process can take anywhere from a few seconds to several hours, depending on the package's size and the speed of your computer.

As with compiling a source RPM, compiling source from a source tarball can fail. This can occur either when configuring the package (via a `configure` script or any other method) or during the actual compilation stage. Diagnosing and fixing such problems requires troubleshooting. Typically, the process fails with an error message shortly before the point at which it stopped. This message should provide you with a clue, but it could be cryptic. If you see many lines of errors, scroll up until you find the first one. This is typically the real problem; subsequent errors occur because of the first error and are not themselves diagnostic.

Once you get the package to compile, you must install it. Most packages today include a way to do this by typing a simple command—typically `./install` or `make install`. This command installs all the package components in their default directories. This process does *not*, though, use your distribution's package management system. If you subsequently install the same package via your distribution's package system, you'll probably end up with two copies of the program. This can cause confusion, because depending on features such as the order of directories in your `PATH` environment variable, either version might be launched. Some packages include a way to uninstall a program to avoid such problems, or simply to remove the package if you decide you don't want it. Typically, you do this by typing `./uninstall` or `make uninstall` in the package directory.

Many packages provide other make targets—that is, commands that can be run with `make`. These may include `make test` (to test that the compile succeeded prior to installing it) and `make clean` (to delete most of the files generated by the compile process to save disk space).

## Special Procedures for the Kernel and Drivers

Kernel compilation is particularly important because the kernel holds most Linux hardware drivers and influences all other aspects of the system. Linux distributions normally ship with kernels that work reasonably well; however, if you want to optimize the system's functioning, one way to do so is to recompile the kernel. In broad strokes, the procedure for doing so is as follows:

1. Obtain kernel source code from <http://www.kernel.org> or some other trusted source.
2. Extract the kernel source code using `tar`. Typically, it resides in a subdirectory of `/usr/src` named after the kernel version, such as `linux-2.6.29` for Linux 2.6.29.
3. Create a symbolic link called `/usr/src/linux` that points to the directory in which the kernel resides.
4. Change into the `/usr/src/linux` directory.
5. Configure the kernel by typing `make config`, `make menuconfig`, or `make xconfig`. This procedure will present you with a huge number of options. Covering them all here is impossible. You'll need to know a lot about your hardware to select the correct options. If in doubt, compile the option as a module, if possible—that will make it available if it's needed but won't increase the size of the main kernel file. It's best if

your kernel contains drivers for your hard disk controller, so be sure to include the appropriate driver.

6. Exit from the configuration utility.
7. Type **make** to build the new kernel. This process is likely to take several minutes and possibly more than an hour.
8. With 2.4.x or earlier kernels, type **make modules**. This command builds the kernel modules (the parts of the kernel it loads from independent files). This action is handled automatically in step 7 with 2.6.x kernels.
9. As root, type **make modules\_install** to install the kernel modules.
10. As root, copy the main kernel file to /boot. The file is stored as **bzImage** in the **arch/i386/boot** subdirectory of the main kernel directory for x86 systems—change **i386** to an appropriate architecture code for other CPUs. I recommend adding the version number to the kernel name, as in **cp arch/i386/boot/bzImage /boot/bzImage-2.6.29**.
11. Add the kernel to your boot loader configuration, as described in Chapter 1. *Do not* replace a working boot loader configuration; *add* the new kernel to your boot loader. This way, you can fall back on your working configuration if there's a problem with the new kernel.
12. Reboot the computer to use the new kernel.

Most modern distributions rely on an *initial RAM disk* to deliver certain drivers to the kernel. These drivers are needed for reading the hard disk, and an initial RAM disk is a RAM-based pseudo-disk that the boot loader delivers to the kernel, enabling the kernel to read the RAM disk files even if it can't yet read the actual hard disk. When you recompile a kernel yourself, it's usually easier to compile the modules into the main kernel file rather than compile them yourself and add them to a RAM disk. If you want or need to build a RAM disk, though, you can do so by using a utility called **mkinitrd**, **mkinitramfs**, or something similar. Details vary from one program or distribution to another, but typically, you'll use it something like this:

```
mkinitramfs -o /boot/initrd-2.6.29.img 2.6.29
```

This command creates an initial RAM disk called **/boot/initrd-2.6.29.img** using the modules associated with the 2.6.29 kernel, which must already be compiled and ready on your system. More advanced configuration options are available, but they require intimate knowledge of the Linux boot process.



You do *not* need to use an initial RAM disk for most kernel drivers, just for drivers that are needed to read files from the hard disk—low-level disk drivers, LVM and RAID drivers, and filesystem drivers. Once the kernel has all the drivers it needs to read your hard disk, it can read additional modules from the hard disk itself.

# Managing Package Dependencies and Conflicts

Although package installation often proceeds smoothly, there are times when it doesn't. The usual sources of problems relate to unsatisfied dependencies or conflicts between packages. The RPM and Debian package management systems are intended to help you locate and resolve such problems, but on occasion (particularly when mixing packages from different vendors), they can actually cause problems. In either event, it pays to recognize these errors and know how to resolve them.



Although dependency and conflict problems are often described in terms of RPM or Debian package requirements, they also occur with tarballs. These more primitive packages lack the means to automatically detect these problems, although some systems, such as Slackware, add dependency checking to their tarballs.

## Real and Imagined Package Dependency Problems

Package dependencies and conflicts can arise for a variety of reasons, including the following:

**Missing libraries or support programs** One of the most common dependency problems is caused by a missing support package. For instance, all KDE programs rely on Qt, a widget set on which these programs are built. If Qt isn't installed, you won't be able to install any KDE packages using RPMs or Debian packages. Libraries are particularly common sources of problems in this respect.

**Incompatible libraries or support programs** Even if a library or support program is installed on your system, it may be the wrong version. For instance, if a program requires Qt 4.5, the presence of Qt 3.2 won't do much good. Fortunately, Linux library naming conventions enable you to install multiple versions of a library, in case you have programs with competing requirements.

**Duplicate files or features** Conflicts arise when one package includes files that are already installed and that belong to another package. Occasionally, broad features can conflict as well, as in two Web server packages. Feature conflicts are usually accompanied by name conflicts. Conflicts are most common when mixing packages intended for different distributions because distributions may split files up across packages in different ways.

**Mismatched names** RPM and Debian package management systems give names to their packages. These names don't always match across distributions. For this reason, if one package checks for another package by name, the first package may not install on another distribution, even if the appropriate package is installed, because that target package has a different name.

Some of these problems are very real and serious. Missing libraries, for instance, must be installed. Other problems, like mismatched package names, are artifacts of the packaging system and are likely symptoms of using packages from a variety of sources. Unfortunately, it's not always easy to tell into which category a conflict fits. When using a package management system, you may be able to use the error message returned by the package system, along with your own experience with and knowledge of specific packages, to make a judgment.

When installing tarballs, and sometimes when compiling a program from source code, you won't get any error messages during installation; you'll see problems only when you try to run the program. These messages may relay an inability to locate a library or run a file, or they may cause the program to crash or otherwise misbehave. Conflicts can be particularly insidious with tarballs because you won't be warned about conflicts, so installing a package can break an existing one, and you might not notice the damage for some time. You can use the `--keep-old-files` qualifier to keep tar from overwriting existing files, though.

## Workarounds to Package Dependency Problems

When you encounter a package dependency or conflict, what can you do about it? There are several approaches to these problems. Some of these approaches work well in some situations but not others, so you should review the possibilities carefully:

**Forcing the installation** One approach is to ignore the issue. Although this sounds risky, in some cases involving failed dependencies, it's appropriate. For instance, if the dependency is on a package that you installed by compiling the source code yourself, you can safely ignore the dependency. You can use rpm's `--nodeps` option to force installation despite dependency problems, or you can use `--force` to override some other issues. With dpkg, `--ignore-depend=package` and `--force-conflicts` have similar effects. Network-enabled meta-packagers make it difficult to use such options and may subsequently complain about any overrides you make.

**Upgrading or replacing the depended-on package** Officially, the proper way to overcome a package dependency problem is to install, upgrade, or replace the depended-on package. If a program requires, say, Qt 4.5 or greater, you should upgrade an older version (such as 4.4) to 4.5. To perform such an upgrade, you'll need to track down and install the appropriate package. This usually isn't too difficult if the new package you want comes from a Linux distribution; the appropriate depended-on package should come with the same distribution.

**Rebuilding the problem package** Some dependencies result from the libraries and other support utilities installed on the computer that compiled the package, not from requirements in the underlying source code. If the software is recompiled on a system that has different packages, the dependencies will change. Therefore, rebuilding a package from source code can overcome at least some dependencies. The `rpmbuild` program does this job for RPM systems, but you'll need to track down a source RPM for the package.



Source packages are also available for Debian systems, but aside from sites devoted to Debian and related distributions, Debian source packages are rare. The sites that do have these packages provide them in forms that typically install easily on appropriate Debian or related systems. For this reason, it's less likely that you'll rebuild a Debian package from source.

**Locating another version of the problem package** Frequently, the simplest way to fix a dependency problem or package conflict is to use a different version of the package you want to install. This could be a newer or older official version (4.2.3 rather than 4.4.7, say), or it might be the same official version but built for your distribution rather than for another distribution.

## Backing Up and Restoring a Computer

Many things can go wrong on a computer that might cause it to lose data. Hard disks can fail, you might accidentally enter some extremely destructive command, a cracker might break into your system, or a user might accidentally delete a file, to name just a few possibilities. To protect against such problems, it's important that you maintain good backups of the computer. To do this, select appropriate backup hardware, choose a backup program, and implement backups on a regular schedule. You should also have a plan in place to recover some or all of your data should the need arise.

### Common Backup Hardware

Just about any device that can store computer data and read it back can be used as a backup medium. The best backup devices are inexpensive, fast, high capacity, and reliable. They don't usually need to be *random-access* devices, though. Random-access devices are capable of quickly accessing any piece of data. Hard disks, USB flash drives, and CD-ROMs are all random-access devices. These devices contrast with *sequential-access* devices, which must read through all intervening data before accessing the sought-after component. Tapes are the most common sequential-access devices. Table 7.10 summarizes critical information about the most common types of backup device. For some, such as tape, there are higher-capacity (and more-expensive) devices for network backups. Numbers are approximate as of mid-2009. Costs are likely to be lower, and capacities higher, in the future.



**TABLE 7.10** Vital Statistics for Common Backup Devices

| Device        | Cost of Drive                                                    | Cost of Media                                  | Uncompressed Capacity | Speed      | Access Type |
|---------------|------------------------------------------------------------------|------------------------------------------------|-----------------------|------------|-------------|
| Tape          | \$200–\$4,000                                                    | \$0.15–\$1.00/GB                               | 24–800GB              | 1–120MB/s  | Sequential  |
| Hard disks    | \$10–\$100<br>(for removable mounting kit or external enclosure) | \$0.15–\$0.50/GB<br>(including mounting frame) | 80–2000GB             | 20–100MB/s | Random      |
| Optical discs | \$25–\$350                                                       | \$0.04–\$0.50/GB                               | 650MB–50GB            | 1–72MB/s   | Random      |

The types of devices that appear in Table 7.10 are those most often used for completely backing up Linux systems. Removable disks (such as USB flash drives or Zip disks) are often used for backing up a few files. Tapes are the traditional backup medium of choice, since they've historically been inexpensive on a per-gigabyte basis. Hard disks and optical discs are now cost-competitive with tape, though. Removable hard disks may provide greater capacity and convenience. These drives may be installed either in special internal drive frames that permit easy swapping or in external drive enclosures that plug into the computer via SCSI, FireWire, or USB ports. Optical discs, depending on the exact format used (CD-R, recordable DVD, or Blu-ray), may be even less expensive but are somewhat lacking in capacity compared to hard disks or tapes.

You may want to design a backup system that uses multiple backup types. For instance, you might perform full backups to a removable hard disk and then perform daily incremental backups onto optical discs. Such a plan can take advantage of the best features of each backup medium.



If you restrict computers' main installation partitions to about 8GB, those entire partitions will most likely fit, when compressed, on standard single-layer recordable DVDs. This can simplify backup and recovery efforts.

It's generally wise to keep multiple backups and to store some of them away from the computers they're meant to protect. Such off-site storage protects your data in case of fire, vandalism, or other major physical traumas. Keeping several backups makes it more likely you'll be able to recover something, even if it's an older backup, should your most recent backup medium fail.

If you decide to use hard disks in removable mounts as a backup medium, you'll need ordinary internal drives and mounting hardware. The hardware comes in two parts: a mounting bay that fits in the computer and a frame in which you mount the hard drive. To use the

system, you slide the frame with hard drive into the mounting bay. You can get by with one of each component, but it's best to buy one frame for each hard drive, which effectively raises the media cost. From a Linux software point of view, removable hard disk systems work like regular hard disks or other removable disk systems, like Zip disks. The disks are likely to be partitioned, and the partitions are likely to hold ordinary Linux filesystems.

Tapes are accessed via special tape device files, such as `/dev/st0` or `/dev/nst0` (for devices that use SCSI drivers) or `/dev/ht0` or `/dev/nht0` (for devices that use PATA drivers). Many backup programs, such as `tar`, can access the tape device file directly.

Optical discs require special programs to write, but they can be mounted much like hard disks. The programs used to write to optical discs are described in the upcoming section “Writing to Optical Discs.”

## Common Backup Programs

Linux supports several backup programs. Some are tools designed to back up individual files, directories, or computers. Others build on these simpler tools to provide network backup facilities. Basic backup programs include `tar` (described earlier in this chapter in “Handling Tarballs”), `dump`, and `cpio`. ARKEIA (<http://www.arkeia.com>) and BRU (<http://www.tolisgroup.com>) are two commercial backup packages that provide explicit network support and GUI front-ends. AMANDA (<http://www.amanda.org>) is a network-capable scripting package that helps `tar` or `dump` perform a backup of an entire network. When dealing with tapes, the `mt` program is useful for controlling the tape hardware. This section provides a look at the `cpio`, `dump`, `restore`, and `mt` tools for performing backups.

### The `cpio` Utility

The `cpio` program is similar in principle to `tar`; it creates an archive file that contains copies of the original files. You can create a `cpio` archive on your hard disk or direct `cpio` to create the archive directly on a tape device. The `cpio` utility has three operating modes:

**Copy-out mode** This mode, activated by using the `-o` or `--create` option, creates an archive and copies files into it.

**Copy-in mode** You activate copy-in mode by using the `-i` or `--extract` option. This mode extracts data from an existing archive. If you provide a filename or a pattern to match, `cpio` will extract only the files whose names match the pattern you provide.

**Copy-pass mode** This mode is activated by the `-p` or `--pass-through` option. It combines the copy-out and copy-in modes, enabling you to copy a directory tree from one location to another.



The names for the copy-out and copy-in modes are confusing.

In addition to the options used to select the mode, `cpio` accepts many other options, the most important of which are summarized in Table 7.11. To back up a computer, you'll

combine the `--create` (or `-o`) option with one or more of the options in Table 7.11; to restore data, you'll do the same but use `--extract` (or `-i`). In either case, `cpio` acts on filenames that you type at the console. In practice, you'll probably use the redirection operator (`<`) to pass a filename list to the program.

TABLE 7.11 Options for use with `cpio`

| Option                               | Abbreviation             | Description                                                                                                                                                                                                                                                        |
|--------------------------------------|--------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>--reset-access-time</code>     | <code>-a</code>          | Resets the access time after reading a file so that it doesn't appear to have been read.                                                                                                                                                                           |
| <code>--append</code>                | <code>-A</code>          | Appends data to an existing archive.                                                                                                                                                                                                                               |
| <code>--pattern-file=filename</code> | <code>-E filename</code> | Uses the contents of <i>filename</i> as a list of files to be extracted in copy-in mode.                                                                                                                                                                           |
| <code>--file=filename</code>         | <code>-F filename</code> | Uses <i>filename</i> as the <code>cpio</code> archive file; if this parameter is omitted, <code>cpio</code> uses standard input or output.                                                                                                                         |
| <code>--format=format</code>         | <code>-H format</code>   | Uses a specified format for the archive file. Common values for <i>format</i> include <code>bin</code> (the default, an old binary format), <code>crc</code> (a newer binary format with a checksum), and <code>tar</code> (the format used by <code>tar</code> ). |
| N/A                                  | <code>-I filename</code> | Uses the specified <i>filename</i> instead of standard input. (Unlike <code>-F</code> , this option does not redirect output data.)                                                                                                                                |
| <code>--no-absolute-filenames</code> | N/A                      | In copy-in mode, extracts files relative to the current directory, even if filenames in the archive contain full directory paths.                                                                                                                                  |
| N/A                                  | <code>-O filename</code> | Uses the specified <i>filename</i> instead of standard output. (Unlike <code>-F</code> , this option does not redirect input data.)                                                                                                                                |
| <code>--list</code>                  | <code>-t</code>          | Displays a table of contents for the input.                                                                                                                                                                                                                        |
| <code>--unconditional</code>         | <code>-u</code>          | Replaces all files, without first asking for verification.                                                                                                                                                                                                         |
| <code>--verbose</code>               | <code>-v</code>          | Displays filenames as they're added to or extracted from the archive. When used with <code>-t</code> , displays additional listing information (similar to <code>ls -l</code> ).                                                                                   |

## Using *cpio* or *tar* to Back Up a Computer

The *cpio* and *tar* commands are generally considered the lowest common denominator backup programs. Tapes created with *cpio* or *tar* can be read on non-Linux systems—something that’s often not true of *dump* archives, whose format is tied to specific filesystems. For this reason, *dump* must explicitly support whatever filesystem you intend to back up. In mid-2009, *dump* supports Linux’s *ext2fs*, *ext3fs*, *ext4fs*, and an XFS-specific *dump* variant is also available, but versions that support other filesystems, such as ReiserFS and JFS, are not yet available.

To compress data, *cpio* and *tar* both rely on an external program, such as *gzip* or *bzip2*, to compress an entire archive. Alternatively, most tape drives support compression in their hardware. Therefore, if your tape drive supports compression, you should *not* compress a *cpio* or *tar* backup. Let the tape drive do that job.

To back up a computer with *cpio*, a command like the following will do the job:

```
find / | cpio -oF /dev/st0
```

Because *cpio* expects a list of files on standard input, this command uses the *find* command and a pipe to feed this information to *cpio*. The *-o* option then tells *cpio* to create an archive, and *-F* specifies where it should be created—in this case, it uses */dev/st0* to create the archive on the tape device.



Both the *find* command and pipes were described in more detail in Chapter 2, “Using Text-Mode Commands.”

This command, though, has some negative effects. Most notably, it backs up everything, including the contents of the */proc* filesystem and any mounted removable disks that might be present. You can use the *-xdev* option to *find* to have that program omit mounted directories from its search, but this means you’ll have to explicitly list each partition you want to have backed up. For instance, you might use a command like the following to back up the */home*, root (*/*), */boot*, and */var* partitions:

```
find /home / /boot /var -xdev | cpio -oF /dev/st0
```



Because tape is a sequential-access medium, the system will restore items in the order in which they were backed up. Therefore, for the fastest partial restores, list the filesystems that you most expect to have to restore first. In this example, */home* is listed first because users sometimes delete files accidentally. Backing up */home* first, therefore, results in quicker restoration of such files.

The procedure for backing up with *tar* is similar; however, *tar* doesn’t need a list of files piped to it; you provide a list of files or directories on the command line:

```
tar cvpf /dev/st0 --one-file-system /home / /boot /var
```

Ordinarily, `tar` descends the directory tree; the `--one-file-system` option prevents this, much like the `-xdev` option to `find`.

After creating a backup with `tar`, you may want to use the `tar --diff` (also known as `--compare`, or `d`) command to verify the backup you've just written against the files on disk. Alternatively, you can include the `--verify (W)` qualifier to have this done automatically. Verifying your backup doesn't guarantee it will be readable when you need it, but it should at least catch major errors caused by severely degraded tapes. On the other hand, the verification will almost certainly return a few spurious errors because of files whose contents have legitimately changed between being written and being compared. This may be true of log files, for instance.



### Real World Scenario

#### Backing Up Using Optical Media

Optical media require special backup procedures. Normally, a program such as `cdrecord` writes to disk using a disc image created by a program like `mkisofs`. This image file is normally an ISO-9660 filesystem—the type of filesystem that's most often found on CD-ROMs.

One option for backing up to optical discs is to use `mkisofs` and then `cdrecord` to copy files to the disc. If you copy files “raw” in this way, though, you'll lose some information, such as write permission bits. You'll have better luck if you create a `cpio` or `tar` file on disk, much as you would when you back up to tape. You would then use `mkisofs` to place that archive in an ISO-9660 filesystem, and then you would burn the ISO-9660 image file to the optical disc. The result will be an optical disc that you can mount and that will contain an archive you can read with `cpio` or `tar`.

A somewhat more direct option is to create an archive file and burn it directly to the optical disc using `cdrecord`, bypassing `mkisofs`. Such a disc won't be mountable in the usual way, but you can access the archive directly by using the CD-ROM device file. On restoration, this works much like a tape restore, except that you specify the CD-ROM device filename (such as `/dev/cdrom`) instead of the tape device filename (such as `/dev/st0`).

### Using *dump* and *restore* to Back Up and Restore a Computer

The `dump` program can create a backup archive of an `ext2`, `ext3`, or `ext4` filesystem, while `xfsdump` performs a similar function for XFS. The `restore` program restores a `dump` archive, while `xfsrestore` does the job for an `xfsdump` archive. These programs all work at a low level compared to `tar` or `cpio`, which means that you can't restore backups across filesystems—a `dump` archive can be restored *only* to an `ext2`, `ext3`, or `ext4` filesystem, not to an XFS, JFS, FAT, or any other type of filesystem.

The syntax for `dump` is as follows:

`dump [options] files`

Table 7.12 summarizes the most important `dump` options. The `xfsdump` program is similar, but some details differ.

**TABLE 7.12** `dump` Options

| Option                           | Description                                                                                                                                                                                                                                   |
|----------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>-level#</code>             | A backup level of 0 signifies a full backup, in which all files are backed up. Higher levels indicate incremental backups, in which only files modified since the last backup of a lower level were backed up.                                |
| <code>-A archive-file</code>     | This option creates a table-of-contents file in the specified file. This can be used to check a dump archive's contents without reading the whole archive.                                                                                    |
| <code>-D file</code>             | This option specifies the location of the file that holds data on incremental backups. The default is <code>/var/lib/dumpdates</code> .                                                                                                       |
| <code>-f file</code>             | You specify a file to hold the backup using this option. You can back up to an ordinary disk file or to a tape device, such as <code>/dev/st0</code> .                                                                                        |
| <code>-jcompression-level</code> | You can specify the amount of compression to apply to the backup with this option, which takes a number as an argument. Note that no space should appear between <code>-j</code> and <code>compression-level</code> . The default value is 2. |
| <code>-Q file</code>             | This option can speed restore times by saving data on where particular files are stored on the tape. The argument is the name of the file that will hold this data.                                                                           |
| <code>-T date</code>             | This option tells <code>dump</code> to back up all files created or modified since the specified date                                                                                                                                         |

As an example of `dump` in action, consider backing up the root (`/`), `/home`, and `/usr` directories to tape. The following command will do this:

```
dump -j5 -f /dev/st0 /home / /usr
```

The `-j5` option specifies moderate compression. Other than this, the command is fairly simple. You can apply additional options from Table 7.12 or from the `dump` man page, of course.

To restore a dump archive, you use the `restore` command. This command can take the `-A`, `-f`, or `-Q` options from Table 7.12. In addition, you must use an option to specify what action you want `restore` to take, as specified in Table 7.13.

TABLE 7.13 `restore` Actions

| Action               | Description                                                                                                                                                                                |
|----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>-C</code>      | Compares archive to files on disk.                                                                                                                                                         |
| <code>-i</code>      | Interactive restore; reads index from the archive and then presents an interactive shell. Consult the <code>restore</code> man page for details on what the commands at this shell permit. |
| <code>-P file</code> | Creates a new access file (similar to the one the <code>-Q</code> dump option creates) from the archive without restoring the contents.                                                    |
| <code>-R</code>      | Restarts an interrupted restore operation.                                                                                                                                                 |
| <code>-r</code>      | Restores data to a pristine (completely empty) filesystem.                                                                                                                                 |
| <code>-t</code>      | Lists the contents of the archive, or of particular files, if they're specified.                                                                                                           |
| <code>-x</code>      | Restores the specified files from the archive.                                                                                                                                             |

### Using *mt* to Control a Tape Drive

In `cpio` and `tar` terminology, each backup is a file. This file is likely to contain many files from the original system, but like an RPM or Debian package file, the archive file is a single entity. Sometimes an archive file is far smaller than the tape on which it's placed. If you want to store more than one archive file on a tape, you can do so by using the nonrewinding tape device file-name. For instance, the following commands store multiple archives on a single tape:

```
tar cvlpf /dev/nst0 /home
tar cvlpf /dev/nst0 /
tar cvlpf /dev/nst0 /boot
tar cvlpf /dev/nst0 /var
```

After you issue these commands, the tape will contain four `tar` files, one for each of the four directories. To access each file after writing them, you need to use a special utility called `mt`. This program moves forward and backward among tape files and otherwise controls tape features. Its syntax is as follows:

```
mt -f device operation [count] [arguments]
```

The *device* parameter is the tape device filename. The `mt` utility supports many operations. Table 7.14 summarizes the most important ones.

**TABLE 7.14** `mt` Operations

| Operation                                    | Description                                                                                                                                                                                      |
|----------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>fsf</code>                             | Moves forward <i>count</i> files.                                                                                                                                                                |
| <code>bsf</code>                             | Moves backward <i>count</i> files.                                                                                                                                                               |
| <code>eod</code> or <code>seod</code>        | Moves to the end of data on the tape.                                                                                                                                                            |
| <code>rewind</code>                          | Rewinds the tape.                                                                                                                                                                                |
| <code>offline</code> or <code>rewoffl</code> | Rewinds and unloads the tape. (Unloading is meaningless on some drives but ejects the tape on others.)                                                                                           |
| <code>retension</code>                       | Rewinds the tape, winds it to the end, and then rewinds it again. This action improves reliability with some types of tape, particularly if the tape has been sitting unused for several months. |
| <code>erase</code>                           | Erases the tape. (This command usually doesn't actually erase the data; it just marks the tape as being empty.)                                                                                  |
| <code>status</code>                          | Displays information on the tape drive.                                                                                                                                                          |
| <code>load</code>                            | Loads a tape into the drive. Unnecessary with many drives.                                                                                                                                       |
| <code>compression</code>                     | Enables or disables compression by passing an argument of 1 or 0, respectively.                                                                                                                  |
| <code>datcompression</code>                  | Also enables and disables compression.                                                                                                                                                           |



The `compression` and `datcompression` operations aren't identical; sometimes a tape drive works with one but not the other.

For instance, suppose you created a backup on a SCSI tape, but now you want to create another backup on the same tape without eliminating the first backup. You could issue the following commands to accomplish this task:

```
mt -f /dev/nst0 rewind
mt -f /dev/nst0 fsf 1
```



```
tar cvlpf /dev/nst0 /directory/to/back/up
mt -f /dev/nst0 offline
```

These commands rewind the tape, space past the first file, create a new backup, and then unload the tape. Such commands are particularly useful when performing incremental backups, as described shortly.

## Performing Network Backups with *rsync*

Some sites implement network-enabled backup tools. Some programs, such as AMANDA, are designed for this purpose; but in many cases simpler tools will do the job, particularly for limited backups, such as backups of user data. One of these simpler tools is *rsync*. This program synchronizes files on one computer with those on another, enabling painless backup of specified directories. You might set up one system with a large hard disk and tape backup drive to hold backups and then have several other computers use *rsync* to copy critical user files onto the backup server. The syntax for the *rsync* client is as follows:

```
rsync [options] user-files destination
```

Table 7.15 summarizes the most important *rsync* options. *user-files* are files to be backed up, and *destination* is a path to the destination. *user-files* or *destination* may be either local files (indicated using a normal Linux pathname) or a remote system (indicated by a leading hostname, and optionally a username, as in `fred@back.luna.edu:/archives`).

**TABLE 7.15** *rsync* Operations

| Operation       | Operation Abbreviation | Description                              |
|-----------------|------------------------|------------------------------------------|
| --archive       | -a                     | Archive mode; equivalent to -rlptgoD.    |
| --recursive     | -r                     | Recurse into subdirectories.             |
| --dirs          | -d                     | Transfers directories without recursing. |
| --links         | -l                     | Copies symbolic links as such.           |
| --copy-links    | -L                     | Copies symbolic links as referent files. |
| --hard-links    | -H                     | Copies hard links as such.               |
| --perms         | -p                     | Preserves permissions.                   |
| --executability | -E                     | Preserves execute permissions.           |
| --owner         | -o                     | Preserves ownership.                     |
| --group         | -g                     | Preserves group ownership.               |

**TABLE 7.15** rsync Operations (continued)

| Operation         | Operation Abbreviation | Description                                                                      |
|-------------------|------------------------|----------------------------------------------------------------------------------|
| --times           | -t                     | Preserves time stamps.                                                           |
| --devices         | none                   | Preserves device files.                                                          |
| --special         | none                   | Preserves special files.                                                         |
| none              | -D                     | Same as --devices --special.                                                     |
| --dry-run         | -n                     | Shows what would happen without actually transferring files.                     |
| --one-file-system | -X                     | Copies from a single filesystem, not from mounted filesystems.                   |
| --compress        | -z                     | Compresses files during transfer. This has no effect on file size on the server. |
| --progress        | none                   | Shows progress indicator during transfer.                                        |

Other protocols may be used for network backups, of course. The File Transfer Protocol (FTP), the Secure Shell (SSH), Network Filesystem (NFS), and others can all handle basic data transfer tasks. The main advantage of rsync is that it transfers only the data that must be transferred to keep the remote archive up-to-date. This fact helps minimize network bandwidth use, which can be an important factor when backing up many or large systems over your network.

## Planning a Backup Schedule

Regular computer backup is important, but precisely *how* regularly is a matter that varies from one system to another. If a computer's contents almost never change (as might be true of a dedicated router or a workstation whose user files reside on a file server), backups once a month or even less often might be in order. For critical file servers, once a day is not too often. You'll have to decide for yourself just how frequently your systems require backup. Consider factors such as how often the data change, the importance of the data, the cost of recovering the data without a current backup, and the cost of making a backup. Costs may be measured in money, your own time, users' lost productivity, and perhaps lost sales.

Even the most zealous backup advocate must admit that creating a full backup of a big system on a regular basis can be a tedious chore. A backup can easily take several hours, depending on backup size and hardware speed. For this reason, most backup packages, including tar, support *incremental backups*. You can create these using the `--listed-incremental file` qualifier to tar. Subsequent backups back up only new or changed files.

You can create a schedule in which you do a full backup of the entire computer only occasionally—say, once a week or once a month. You’d do this by deleting the increment file and running a backup as usual. On intervening weeks or days, you can perform an incremental backup, thus saving time.

With `cpio`, the key to incremental backups is in the list of files fed to the program. You can perform an incremental backup by using `find` options to locate only new files or files that have changed since the last backup. For instance, the `-newer file` option to `find` causes that program to return only files that have been modified more recently than `file`. Thus, you could create a file (perhaps a log of your backup activity) during each backup and use it as a way of determining what files have been modified since the last backup.

Performing incremental backups has a couple of drawbacks. One is that they complicate restoration. Suppose you do a full backup on Monday and incremental backups every other day. If a system fails on Friday, you’ll need to restore the full backup and several incremental backups. Second, after restoring an incremental backup, your system will contain files that you had deleted since the full backup. If files have short life spans on a computer, this can result in a lot of “dead” files being restored when the time comes to do so.

Despite these problems, incremental backups can be extremely useful for helping make backups manageable. They can also reduce wear and tear on tapes and tape drives, and they can minimize the time it takes to restore files if you know that the files you need to restore were backed up on an incremental tape.

## Preparing for Disaster: Backup Recovery

Creating backups is advisable, but doing this isn’t enough. You must also have some way to restore backups in case of disaster. This task involves two aspects: partial restores and emergency recovery.

Partial restores involve recovering just a few noncritical files. For instance, users might come to you and ask you to restore files from their home directories. You can do so fairly easily by using the `--extract (x)` `tar` command:

```
cd /
tar xvpf /dev/st0 home/username/filename
```



This sequence involves changing to the root directory and issuing a relative path to the file or directory that must be restored. This is required because `tar` normally strips away the leading `/` in files it backs up, so the files are recorded in the archive as relative filenames. If you try to restore a file with an absolute filename, it won’t work.

When you’re using `cpio`, the procedure is similar, but you use the `--extract (-i)` option, along with other options to feed the name of the archive, and perhaps do other things:

```
cd /
cpio -ivF /dev/st0 home/username/filename
```

Whether you're using `tar` or `cpio`, you'll need to know the exact name of the file or directory you want to restore in order to do this. If you don't know the exact filename, you may need to use the `--list (t)` command to `cpio` or `tar` to examine the entire contents of the tape, or at least everything until you see the file you want to restore.



If you use incremental backups, you can use the incremental file list to locate the filename you want to restore.

A much more serious problem is that of recovering a system that's badly damaged. If your hard disk has crashed or your system has been invaded by crackers, you must restore the entire system from scratch, without the benefit of your normal installation. In most cases, an emergency system, as described in Chapter 1, will do the job. You'll need to ensure beforehand that your emergency system has the tools you need to access your backups.

Whatever approach you choose to use, you should test it before you need it. Learn at least the basics of the tools available in any system you plan to use. If you use unusual backup tools (such as commercial backup software), be sure to copy those tools to your emergency system or have them available on a separate removable disk. If you'll need to recover clients via network links, test those setups as well.

You may not be able to *completely* test your emergency restore tools. Ideally, you should boot the tools, restore a system, and test that the system works. This may be possible if you have spare hardware on which to experiment, but if you lack this luxury, you may have to make do with performing a test restore of a few files and testing an emergency boot procedure. Note that a freshly restored system will not be bootable; you'll need a way to restore your boot loader from an emergency boot.

## Writing to Optical Discs

Optical media are an extremely popular means of exchanging moderately large files. Most CD-R and CD-RW media hold 700MB of files (older discs held 650MB), recordable DVD formats have capacities of 4.7–8.5GB, and Blu-ray discs hold 25–50GB. Plain write-once CDs and DVDs cost \$0.10 to \$1 and are likely to remain readable for several years to decades, given proper storage. You can't simply mount an optical disc and write files to it as you would a floppy disk, though; you must create a complete filesystem and then copy (or “burn”) that filesystem to the disc. This process requires using two tools, `mkisofs` and `cdrecord`, or requires variants of or front-ends to these tools.

### Linux Optical Disc Tools

The Linux optical disc creation process involves three steps:

1. Collect source files. You must first collect source files in one location, typically a single subdirectory of your home directory.

2. Create a filesystem. You point a filesystem-creation program, `mkisofs`, at your source directory. This program generates an ISO-9660 filesystem in an image file. Alternatively, you can create another filesystem in an appropriately sized partition or image file and copy files to that partition or image file. This latter approach can be used to create ext2fs, FAT, or other types of optical discs, but there's seldom any advantage to doing this.



If you install an OS to a partition that's less than your media's size, you can back it up by burning the partition directly to CD-R or recordable DVD. The result is a disc that uses the OS's native filesystem. You can restore the backup by using `dd`, assuming the target partition is exactly the same size as the original.

3. Burn the disc. You use an optical disc-burning program, such as `cdrecord`, to copy the image file to the optical device.



Recent Linux distributions provide both `mkisofs` and `cdrecord` in a single package called `cdrtools`.

The `growisofs` program combines the functionality of `mkisofs` and `cdrecord`, but `growisofs` works only with DVDs and Blu-ray discs, not with the smaller CD-Rs. In turn, many versions of `cdrecord` won't work with the larger DVDs and Blu-ray discs!

Another approach to optical disc creation is to use GUI front-ends to the text-mode tools. These GUI tools provide a point-and-click interface, eliminating the need to remember obscure command-line parameters. Popular GUI Linux optical disc creation tools include X-CD-Roast (<http://www.xcdroast.org>), GNOME Toaster (<http://gnometoaster.rulz.org>), and K3B (<http://k3b.sourceforge.net>).

All of these optical disc tools provide a dizzying array of options. For the most part, the default options work quite well, although you will need to provide information to identify your drive and burn speed, as described in the next section. Some `mkisofs` options can also be important in generating image files that can be read on a wide variety of OSs, as described later in "Creating Cross-Platform Discs."

## A Linux Optical Disc Example

To begin creating optical discs, starting with `mkisofs` makes sense:

```
$ mkisofs -J -r -V "volume name" -o ../image.iso ./
```

This command creates an image file called *image.iso* in the parent of the current directory, placing files from the current working directory (`./`) in the resultant image file. The `-J` and `-r` options enable Joliet and Rock Ridge extensions, respectively, and the `-V` option sets the volume name to whatever you specify. Dozens of other options and variants on these are available; check the `mkisofs` man page for details.

Once you've created an image file, you can burn it with a command such as the following:

```
$ cdrecord dev=/dev/dvdrw speed=4 ../image.iso
```

The device (dev=/dev/dvdrw) must exist and be your optical drive. (This may be /dev/dvdrw or something similar even if you're burning a CD-R. Details vary depending on your distribution and hardware.) The speed is set using the `speed` option, and the final parameter specifies the source of the file to be burned. As with `mkisofs`, `cdrecord` supports many additional options; consult its man page for details. If the SUID bit isn't set on this program, with ownership set to `root`, you must run it as `root`.



You can use the loopback option to verify the contents of an image file before burning it. For instance, typing `mount -t iso9660 -o loop image.iso /mnt/cdrom` mounts the `image.iso` file to `/mnt/cdrom`. You can then check that all the files that should be present are present. You must be `root` to use this option, or you must have created an appropriate `/etc/fstab` entry.

When burning DVDs or Blu-ray discs, you may need to use `growisofs`, which combines the features of both `mkisofs` and `cdrecord`:

```
$ growisofs -speed=4 -Z /dev/dvdrw -J -r -V "volume name" ./
```

The `-speed` option of `growisofs` is equivalent to the `speed` option of `cdrecord`. You specify the target device using `-Z` rather than `dev=`. Options following the device are the same as those used by `mkisofs`. The `growisofs` approach eliminates the need for a temporary image file, which is particularly helpful with larger discs. If you prefer, though, you can create such a file with `mkisofs` or some other utility and then burn it with `growisofs` by adding the source file to the `-Z` option:

```
$ growisofs -speed=4 -Z /dev/dvdrw=source-file.iso
```

## Creating Cross-Platform Discs

You may want to create a disc that works on many different OSs. If so, you may want to use a wide range of filesystems and filesystem extensions. Such discs contain just one copy of each file; the filesystems are written in such a way that they all point their unique directory structures at the same files. Thus, the extra space required by such a multiplatform disc is minimal. Features you may want to use on such a disc include the following:

**Following symbolic links** The `-f` option to `mkisofs` causes the tool to read the files that symbolic links point to and include them on the CD-R, rather than to write symbolic links as such using Rock Ridge extensions. Following symbolic links can increase the disk space used on a CD-R, but this option is required if you want symbolic links to produce reasonable results on systems that don't understand Rock Ridge, such as Windows.

**Long ISO-9660 filenames** Normally, `mkisofs` creates only short filenames for the base ISO-9660 filesystem. Long filenames are stored in Rock Ridge, Joliet, or other filesystem extensions. You can increase the raw ISO-9660 name length to 31 characters with the `-l` (that's a lowercase *L*) option.

**Joliet support** The `-J` option to `mkisofs`, as noted earlier, creates an image with Joliet extensions. These extensions do *not* interfere with reading the disc from OSs that don't understand Joliet.

**Rock Ridge support** The `-R` and `-r` options both add Rock Ridge extensions. The `-R` option adds the extensions, but it doesn't change ownership or permissions on files. Using `-r` works the same, except that it changes ownership of all files to `root`, gives all users access to the files, and removes write permissions. These features are usually desirable on a disc that's to be used on any but the original author's computer.

**UDF support** You can add support for the Universal Disk Format (UDF) filesystem by including the `-udf` option. UDF is the “up and coming” optical disc filesystem and is the preferred filesystem for DVDs. Most modern OSs, including recent versions of Linux, Windows, and Mac OS, understand UDF.

**HFS support** To create a disc that includes Mac OS HFS support, add the `-hfs` option. When you insert the resulting disc into a Macintosh, the computer will read the HFS filenames. A slew of options are related to this one.

## Summary

One of your primary duties as a system administrator is to manage the packages installed on a computer. To do this, you must often remove unused programs, install new ones, and upgrade existing packages. You may also need to verify the integrity of installed programs or track down what libraries or other programs another one uses. In all these tasks, the RPM and Debian package management systems can be extremely helpful. These systems track installed files and dependencies, giving you access to information that's not otherwise available. On occasion, though, you may need to use the simpler tarballs—particularly if you use a tarball-based distribution such as Slackware.

Backup is critically important for most computers, but backup is also often neglected. Traditionally, tapes have been used to back up computers, but the cost of hard disks has dropped so much that removable disks are now a viable alternative for many installations. Typically, systems are backed up using tools designed for this purpose, such as `tar`, `cpio`, or `dump`. Such programs can write directly to tape devices, or they can be used to create archive files on removable disks. You can also create an archive file that's subsequently stored on an optical disc using `cdrecord`.

Optical media can be very convenient, but they require special tools to be created. The `mkisofs` program creates a filesystem for such media, while `cdrecord` stores the filesystem on disk. The `growisofs` program combines these two tools' functionality into one program. GUI front-ends to these tools, such as X-CD-Roast, can simplify the creation of optical discs.

## Exam Essentials

**Identify critical features of RPM and Debian package formats.** RPM and Debian packages store all files for a given package in a single file that also includes information on what other packages the software depends on. These systems maintain a database of installed packages and their associated files and dependencies.

**Describe the process of installing an RPM or Debian package.** Use the `rpm` program or `Yum` to install an RPM package, or use `dpkg` or `apt-get` to install a Debian package. These programs install, upgrade, or remove all files associated with a package and maintain the associated databases.

**Summarize methods of working around package dependency problems.** Dependency problems can be overcome by forcing an installation, upgrading or installing the depended-on package, recompiling the package, or installing another version of the target package. Which approach is best depends on the specifics of the system involved.

**Describe how to install a program from a source code tarball.** Compiling a program from source code depends greatly on the program in question. Most provide a configuration script called `configure` or a `configure` target in the `Makefile`. Once that's run, you type **make** to build the package and then install it with an `install` script or an `install` target in the `Makefile`.

**Summarize backup hardware options.** Backup hardware includes tapes, dedicated hard disks, removable disks, and optical media. Tapes have been the most common type of backup hardware in the past, but each of the others has its place for particular backup types, and hard disks have dropped in price enough to make them appealing as an everyday backup medium.

**Describe how Linux writes to optical media.** Linux uses the `mkisofs` program to create an ISO-9660 filesystem (and optionally other common optical disc filesystems), which is then burned to the disc by `cdrecord`. These programs may be piped together, and common GUI front-ends can help in this process by providing a friendlier user interface.



## Review Questions

1. You are installing a small program from source code and need to change a number of options. You cannot find a specific configuration script for the program. In this case, what file should you edit?
  - A. `configfile`
  - B. `change`
  - C. `make`
  - D. `makefile`
2. Which of the following is *not* an advantage of a source package over a binary package?
  - A. A single source package can be used on multiple CPU architectures.
  - B. By recompiling a source package, you can sometimes work around library incompatibilities.
  - C. You can modify the code in a source package, altering the behavior of a program.
  - D. Source packages can be installed more quickly than binary packages can.
3. Which is true of using both RPM and Debian package management systems on one computer?
  - A. It's generally inadvisable because the two systems don't share installed file database information.
  - B. It's impossible because their installed file databases conflict with one another.
  - C. It causes no problems if you install important libraries once in each format.
  - D. It's a common practice on Red Hat and Debian systems.
4. Which of the following statements is true about binary RPM packages that are built for a particular distribution?
  - A. They can often be used on another RPM-based distribution for the same CPU architecture, but this isn't guaranteed.
  - B. They may be used in another RPM-based distribution only when you set the `--convert-distrib` parameter to `rpm`.
  - C. They may be used in another RPM-based distribution only after you convert the package with `alien`.
  - D. They can be recompiled for an RPM-based distribution running on another type of CPU.
5. Which is true of source RPM packages?
  - A. They consist of three files: an original source tarball, a patch file of changes, and a PGP signature indicating the authenticity of the package.
  - B. They require programming knowledge to rebuild.
  - C. They can sometimes be used to work around dependency problems with a binary package.
  - D. They are necessary to compile software for RPM-based distributions.

6. Which of the following do RPM filenames conventionally include?
  - A. Single-letter codes indicating Red Hat–certified build sites
  - B. Build date information
  - C. Version number and CPU architecture information
  - D. A CRC code for the package’s contents
7. To use `dpkg` to remove a package called `theprogram`, including its configuration files, which of the following commands would you type?
  - A. `dpkg -P theprogram`
  - B. `dpkg -p theprogram`
  - C. `dpkg -r theprogram`
  - D. `dpkg -r theprogram-1.2.3-4.deb`
8. Which of the following describes a difference between `apt-get` and `dpkg`?
  - A. `apt-get` provides a GUI interface to Debian package management; `dpkg` does not.
  - B. `apt-get` can install tarballs in addition to Debian packages; `dpkg` cannot.
  - C. `apt-get` can automatically retrieve and update programs from Internet sites; `dpkg` cannot.
  - D. `apt-get` is provided only with the original Debian distribution, but `dpkg` comes with Debian and its derivatives.
9. Which of the following is true of an attempt to use a Debian package from one distribution on another Debian-derived distribution?
  - A. It’s unlikely to work because of library incompatibilities and divergent package-naming conventions.
  - B. It’s guaranteed to work because of Debian’s strong package definition and enforcement of standards for startup scripts and file locations.
  - C. It will work only when the distributions are built for different CPUs or when the `alien` package is already installed on the target system.
  - D. It’s likely to work because of the close relationship of Debian-based distributions, assuming the two distributions are for the same CPU architecture.
10. The `tar` program may be used to complete which of the following tasks? (Choose all that apply.)
  - A. Install RPM and Debian packages.
  - B. Install software from binary tarballs.
  - C. Back up a computer to tape.
  - D. Create source code archive files.

11. The `tar` program provides a much easier \_\_\_\_\_ process than RPM and Debian package tools do.
  - A. dependency tracking
  - B. source code compilation
  - C. file ownership setting
  - D. package creation
  
12. What is wrong with the following commands, which are intended to record an incremental backup on a tape that already holds one incremental backup?
 

```
mt -f /dev/st0 fsf 1
tar cvlpf /dev/st0 --listed-incremental /root/inc /home
```

  - A. The `mt` command should terminate in 2, rather than 1, to skip to the second position on the tape.
  - B. When backing up `/home`, the incremental file must reside in `/home`, not in `/root`.
  - C. The device filename should be a nonrewinding name (such as `/dev/nst0`), not a rewinding name (`/dev/st0`).
  - D. The incremental backup must include the root (`/`) directory; it cannot include only `/home`.
  
13. What is an advantage of using `rsync` for network backups, as compared to FTP or NFS?
  - A. `rsync` can be used by any user.
  - B. `rsync` can reduce network bandwidth use.
  - C. `rsync` can back up across architectures (x86 to PowerPC, for instance).
  - D. `rsync` can back up from any filesystem type (XFS, ReiserFS, and so on).
  
14. Which of the following is a true statement concerning the ability to restore a backup used making Linux's `dump` to a partition that's formatted with XFS?
  - A. You must use `cpio`'s `-H dump` option to do the job.
  - B. You must use the `dump` utility's `-r` option to do the job.
  - C. You must use the `xfsrestore` utility to do the job.
  - D. This task cannot be directly and easily performed.
  
15. You need to restore some files that were accidentally deleted. Which of the following commands can be used to list the contents of an archive stored on a SCSI tape?
  - A. `cpio -itv > /dev/st0`
  - B. `cpio -otv > /dev/st0`
  - C. `cpio -otv < /dev/st0`
  - D. `cpio -itv < /dev/st0`

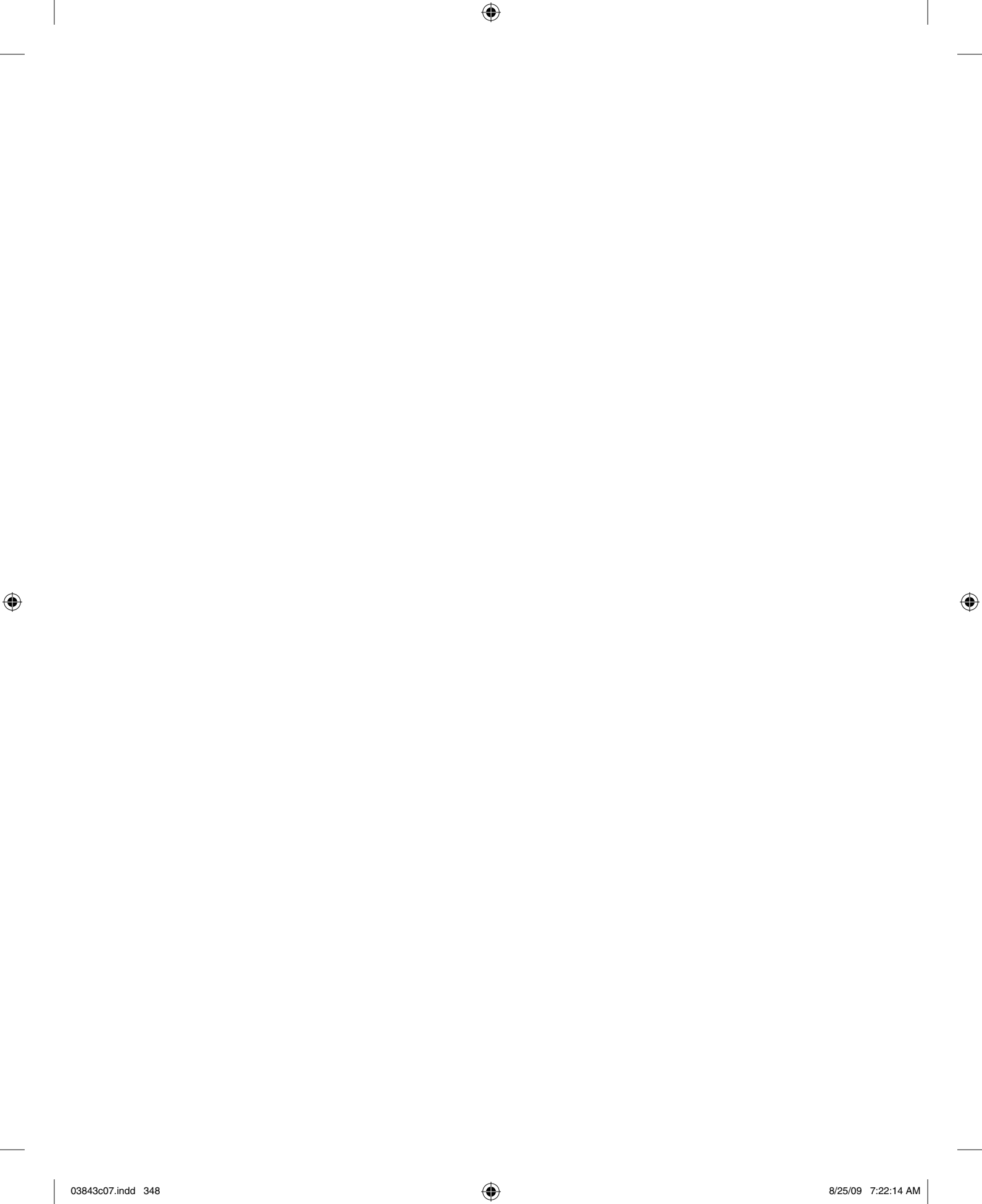
16. You arrive at work on Monday morning to find that the server has crashed. All indications point to the crash as occurring after midnight on Monday morning. Scripts automatically do a full backup of the server every Friday night and an incremental backup all other nights. Which tapes do you need to restore the data on a new server? (Choose all that apply.)
  - A. Thursday's tape
  - B. Friday's tape
  - C. Saturday's tape
  - D. Sunday's tape
17. What is a disadvantage of optical discs compared to tapes as a backup medium?
  - A. Optical discs are sequential-access media; tapes are random access.
  - B. Optical discs require use of the awkward `mt` utility.
  - C. Optical discs are much more expensive than tapes.
  - D. Optical discs have lower capacity than tapes.
18. You create a backup of important files on CD-R by creating an uncompressed tarball and then writing that tarball to disc by using `cdrecord`. How would you access these files in the future?
  - A. Mount the disc as an ordinary CD-R using `mount`, and then read from the tarball file on the mount point, as in `tar xvf /mnt/dvd/backup.tar`.
  - B. Pass the optical disc's device filename to `tar` as the input file, as in `tar xvf /dev/dvd`.
  - C. Dynamically copy the optical disc to a tape using the `dco` command, and then access the tape device using `tar`, as in `tar xvf /dev/st0`.
  - D. Use `mkisofs` to generate a loopback filesystem and mount it at `/mnt/dvd`, and then access it with `tar`, as in `tar xvf /mnt/dvd/backup.tar`.
19. What option to `mkisofs` would you use if you want a computer running Microsoft Windows Vista to be able to read long filenames on a CD-R or DVD created with Linux? (Choose all that apply.)
  - A. `-J`
  - B. `-r`
  - C. `-hfs`
  - D. `-udf`
20. You've downloaded the latest version of your Linux distribution as a 4GB DVD image file (`distrib.iso`). Which of the following commands will burn this file to a blank DVD, assuming your DVD drive can be accessed as `/dev/dvdrw`?
  - A. `growisofs -Z /dev/dvdrw distrib.iso`
  - B. `cdrecord -Z /dev/dvdrw distrib.iso`
  - C. `growisofs -Z /dev/dvdrw=distrib.iso`
  - D. `mkisofs -o /dev/dvdrw -i distrib.iso`

## Answers to Review Questions

1. D. Some programs (particularly small ones) don't use configuration scripts. To change their options, you must typically edit a file called `Makefile` or `makefile`.
2. D. Because they must be compiled prior to installation, source packages require *more* time to install than binary packages do.
3. A. Package management systems don't share information, but neither do their databases actively conflict. Installing the same libraries using both systems would almost guarantee that the files served by both systems would conflict with one another. Actively using both RPM and Debian packages isn't common on any distribution, although it's possible with all of them.
4. A. RPMs are usually portable across distributions, but occasionally they contain incompatibilities. There is no `--convert-distrib` parameter to `rpm`, nor is `alien` used to convert from RPM format to RPM format. Binary packages can't be rebuilt for another CPU architecture, but source packages may be rebuilt for any supported architecture, provided the source code doesn't rely on any CPU-specific features.
5. C. Some dependencies result from dynamically linking binaries to libraries at compile time, and so they can be overcome by recompiling the software from a source RPM. Option A describes Debian source packages, not RPM packages. Recompiling a source RPM requires only issuing an appropriate command, although you must also have appropriate compilers and libraries installed. Source tarballs can also be used to compile software for RPM systems, although this results in none of RPM's advantages.
6. C. The package version number (as well as an RPM build number) and CPU architecture code (or `src` for source code or `noarch` for architecture-independent files) are included in most RPM package filenames. Red Hat does not provide certification for RPM maintainers. Build dates are stored in the RPM, but not in the filename. CRC codes also don't conventionally appear in RPMs' filenames.
7. A. An uppercase `-P` invokes the purge operation, which completely removes a package and its configuration files. The lowercase `-p` causes `dpkg` to print information on the package's contents. The `-r` parameter removes a package but leaves configuration files behind. The final variant (option D) also specifies a complete filename, which isn't used for removing a package—you should specify only the shorter package name.
8. C. You can specify Debian package archive sites in `/etc/apt/sources.list`, and then you can type **`apt-get update`** and **`apt-get upgrade`** to quickly update a Debian system to the latest packages. GUI package management tools for Debian and related distributions exist, but they aren't `apt-get`. Neither `apt-get` nor `dpkg` can directly install tarballs. Both Debian and many of its derivatives ship with both `apt-get` and `dpkg`.

9. D. Systems that use Debian are based on the same core OS, and so they share most components, making package transplants likely—but not certain—to succeed. Library incompatibilities *could* cause problems but aren't likely to, especially if you use recent packages and distributions. Although Debian has clearly defined key file locations, startup scripts, and so on, these can't guarantee success. Binary packages built for *different* CPUs are almost guaranteed *not* to work, although scripts or other nonbinary packages most likely will work across CPU types.
10. B, C, D. The `tar` program can do all these things except for directly installing RPM or Debian packages.
11. D. The `tar --create` command creates an archive from any specified directory; RPM and Debian package creation tools are more complex than this. The `tar` utility provides no dependency-tracking mechanisms at all, making you do that work. Although `tar` can be used to distribute source code, it's not used in compiling it per se. All the package tools discussed in this chapter automatically set file ownership appropriately.
12. C. The `/dev/st0` device (and `/dev/ht0`, for that matter) rewinds after every operation. Therefore, the first command as given will wind past the first incremental backup, and then immediately rewind. The second command will therefore overwrite the first incremental backup.
13. B. `rsync` was designed to minimize network bandwidth use by figuring out which files already exist on a backup server and therefore don't need to be transferred. Thus, option B describes an advantage of this tool for network backups. Options A, C, and D correctly describe `rsync` features; however, they also describe features shared by FTP, NFS, and many other tools that can be used for network backup, so these aren't advantages of `rsync`.
14. D. The `dump` utility is filesystem-specific, so restores can be performed only to the same filesystem format that was used on the source system. Thus, restoring a backup from `ext2fs`, `ext3fs`, or `ext4fs` (the three filesystems supported by `dump`) to XFS is impossible, at least directly, as option D states. (You could create a carrier filesystem image file, restore the data to it, and then copy the files out of that file, but this is indirect and awkward, not direct and easy.) Although `cpio` has a `-H` option that permits it to read various archive types, as specified by option A, `dump` is not a supported value. Contrary to option B, `dump` is not used to restore data. The `xfsrestore` program, mentioned in option C, exists, but it can read only those archives created by `xfsdump`, not by `dump`.
15. D. With the `cpio` utility, the `-i` option is used to read in from an external source—in this case coming in (`<`) from `/dev/st0`. The `-tv` options are used to show the files on the tape and provide a listing of what is there.
16. B, C, D. To restore the data, you must restore the most recent full backup—which was done on Friday night. After the full restore, you must restore the incremental backups in the order in which they were done. In this case, two incrementals (Saturday's and Sunday's) were done after the full backup, and they must be restored as well.

17. D. The highest-capacity optical discs are dual-layer Blu-ray discs, which can store 50GB per disc. Tapes can store up to 800GB per tape. (These values are likely to increase in the future for both storage types.) Option A is backward; tapes are the sequential-access devices, and optical discs are random access. Option B is also backward, and it's deceptive; `mt` is a tape control utility, and it's not particularly awkward. The relative costs of optical discs and tapes are similar; they might favor one or the other depending on specific products chosen, but contrary to option C, optical discs are not universally or significantly more expensive than tapes.
18. B. The question describes writing a tarball "raw" to the CD-R, without the benefit of a filesystem. When written in this way, the tarball may be accessed in an equally "raw" way, as described by option B. Option A describes the process you'd use if you'd created a filesystem to hold the tarball (via `mkisofs`) and written that to disc rather than writing the tarball directly. There is no standard `dco` command, so option C won't work. Option D is, in a sense, backward; you might use `mkisofs` to create a filesystem *before* writing the backup to disc but not after doing so.
19. A, D. The `-J` option creates a disc with Joliet extensions, and `-udf` creates one with the UDF filesystem. Recent versions of Windows understand both of these extensions. The `-r` option creates a disc with Rock Ridge extensions, while `-hfs` creates one with an Apple HFS filesystem. Windows won't understand either of these without special drivers, although Windows will still be able to read the underlying ISO-9660 filesystem (with 8.3 filenames).
20. C. The `growisofs` program is generally used to burn DVDs, since many versions of `cdrecord` lack DVD support. Of the two `growisofs` command options, C presents the correct syntax; option B is missing a critical equal sign (=) between the device filename and the image filename. Even if your version of `cdrecord` supports DVDs, option B's syntax is incorrect; `cdrecord` uses `dev=` rather than `-Z` to specify the target device. The `mkisofs` command is used to create an image file, not to burn one to disc; and option D's syntax is incorrect in any event.





# Chapter 8

## Configuring Basic Networking

---

### THE FOLLOWING COMPTIA OBJECTIVES ARE COVERED IN THIS CHAPTER:

- ✓ **4.2 Execute network interface configuration using the following** (dhclient, dhcpd, ifconfig, iwconfig, route, ifup, ifdown, **network configuration files**).
- ✓ **4.3 Implement configurations and/or configuration changes for the following** (Hostname lookup: /etc/hosts, /etc/nsswitch.conf, /etc/resolv.conf).
- ✓ **4.5 Troubleshoot basic connectivity issues using the following tools** (netstat, ping, traceroute, arp, telnet, route).
- ✓ **4.6 Troubleshoot name resolution issues using the following tools** (dig, host, nslookup, hostname).



Networking is a complex topic that's touched on in several chapters of this book. This chapter provides an introduction to basic Transmission Control Protocol/Internet Protocol (TCP/IP) network configuration. Although most users perceive network connections as things that “just work” (or “just *don't* work” on bad days!), the reality is that even the basics of network configuration require a certain amount of knowledge. You must understand the roles of several different types of network addressing. Several tools can be involved in network configuration, including both tools to manage the task manually and with the help of a network server. When things go wrong, you must be able to resolve the problems, so this chapter also covers network troubleshooting tools and techniques.



Objective 4.3 is covered partly in this chapter and partly in Chapter 9.

## Understanding Networks

In the past three decades, computer networks have grown dramatically. Both local networks and larger networks exploded in importance as increasingly sophisticated network applications were written. To understand these applications, it's useful to know something about network hardware and the most common network protocols. Both of these things influence what a network can do.

### Basic Functions of Network Hardware

Network hardware is designed to enable two or more computers to communicate with one another. As described shortly, this hardware can take a variety of forms. Most commonly, network hardware comes as a card you plug into a computer, as a device that plugs into a USB or other external port, or as circuitry that's built into a computer motherboard. Many networks rely on wires or cables to transmit data between machines as electrical impulses, but network protocols that use radio waves or even light to do the job are also common.

Sometimes the line between network hardware and peripheral interface ports can be blurry. For instance, a parallel port is normally not considered a network port, but when it is used with the Parallel Line Interface Protocol (PLIP; <http://tldp.org/HOWTO/PLIP.html>), the

parallel port becomes a network device. More commonly, a USB or RS-232 serial port can become a network interface when used with the Point-to-Point Protocol (PPP), which enables network connections over these ports, typically used with a modem over a telephone line.



PPP was once an important networking tool, but with the rise of broadband Internet connections, it's falling in importance. Some digital subscriber line (DSL) connections still use a PPP variant, called PPP over Ethernet (PPPoE). Linux can handle PPPoE, or you can use a broadband router to make the PPPoE connection, and Linux can then use an ordinary Ethernet link to the broadband router.

At its core, network hardware is hardware that facilitates the transfer of data between computers. Hardware that's most often used for networking includes features that help this transfer in various ways. For instance, such hardware may include ways to address data intended for specific remote computers, as described later in the section "Hardware Addresses." When basically non-network hardware is pressed into service as a network medium, the lack of such features may limit the utility of the hardware or require extra software to make up for the lack. If extra software is required, you're unlikely to notice the deficiencies as a user or system administrator because the protocol drivers handle the work, which makes them harder to configure and more prone to sluggishness or other problems.

## Types of Network Hardware

Aside from traditionally non-network ports like USB, RS-232 serial, and parallel ports, Linux supports several types of common network hardware:

**Ethernet** Ethernet is the most common type of network hardware on local networks today. It comes in several varieties ranging from the old 10Base-2 and 10Base-5 (which use coaxial cabling similar to cable TV cable) to 100Base-T and 1000Base-T (which use twisted-pair cabling that resembles telephone wire but with broader connectors, which is the reason for the "-T") to the cutting-edge 10GBase-LR, 10GBase-SR, and other variants. The number preceding the "Base" (short for "baseband," a type of transmission medium) indicates the technology's maximum speed in megabits per second (Mbps). The *G* in *10GBase* refers to a speed in gigabits per second (Gbps). (1000Base-T is also referred to as *gigabit Ethernet*). Of the versions in use in 2009, 100Base-T and 1000Base-T are the most common for existing installations, but the latter is nearly universally supported in new hardware. Linux includes excellent Ethernet support, including drivers for almost every Ethernet card on the market.

**Token Ring** At one time an important competitor to Ethernet, IBM's *Token Ring* technology is today fairly uncommon. Speeds range from 4Mbps to 1Gbps. Token Ring is costlier than Ethernet and has less in the way of hardware support. For instance, fewer printers support direct connection to Token Ring networks than to Ethernet networks. Linux includes support for several Token Ring cards, so if you need to connect Linux to a Token Ring network, you can do so.

**FDDI** *Fiber Distributed Data Interface (FDDI)* is a networking technology that's comparable to 100Base-T Ethernet in speed. FDDI uses fiber-optic cables, but a variant known as CDDI works over copper cables similar to those of 100Base-T. Both technologies are supported by the Linux FDDI drivers.

**HIPPI** *High-Performance Parallel Interface (HIPPI)* provides 800Mbps or 1600Mbps speeds. It's most commonly used to link computer clusters or supercomputers over dozens or hundreds of meters. Linux includes limited HIPPI support.

**LocalTalk** *LocalTalk* is a network hardware protocol developed by Apple for its Macintosh line. It's slow by today's standards (2Mbps), and Apple no longer includes LocalTalk connectors on modern Macintoshes.

**Fibre Channel** *Fibre Channel* supports both optical and copper media, with speeds of between 133Mbps and 6800Mbps. The potential reach of a Fibre Channel network is unusually broad—up to 10 kilometers. Linux includes support for some Fibre Channel products.

**Wireless protocols** Several wireless networking standards (often called *Wi-Fi*) have become popular, particularly in small offices, homes, and public areas. These protocols vary in speed and range. The most popular of these standards are 802.11b, which supports operation at up to 11Mbps, and 802.11g, which works at up to 54Mbps. The 802.11a standard also works at up to 54Mbps but is less common. The 802.11n standard is not yet finalized but is expected to support speeds of up to 600Mbps. Several products with speeds in between those of 802.11g and 802.11n are in production, with varying degrees of standardization. This area is evolving rapidly, so you should be particularly careful about checking on Linux driver availability for any wireless networking products you buy.

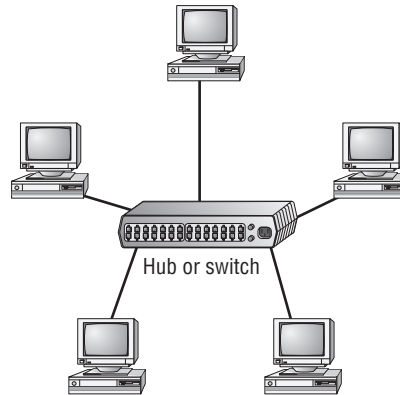
If you're putting together a new network for a typical office, chances are that 100Base-T or gigabit Ethernet is the best choice. Wireless products are a good choice if running new cabling is a hassle and speed isn't vitally important or if you want to provide a network that enables roaming use of notebook computers. If you need to connect to an existing network, you should find out what type of hardware it uses. If necessary, consult with your local network administrator to find out what type of network card you require.



Most modern computers ship with network hardware preinstalled. This hardware is almost always Ethernet, although notebooks usually include Wi-Fi hardware, as well.

In addition to the network cards you place in your computers, you need network hardware outside of the computer. With the exception of wireless networks, you'll need some form of network cabling that's unique to your hardware type. (For 100Base-T Ethernet, get cabling that meets at least Category 5, or Cat-5, specifications. For 1000Base-T Ethernet, get Cat-5e or better cabling.) Many network types, including twisted-pair Ethernet, require the use of a central device known as a *hub* or *switch*. You plug every computer on a local network into this central device, as shown in Figure 8.1. The hub or switch then passes data between the computers.

**FIGURE 8.1** Many networks link computers together via a central device known as a hub or switch.



As a general rule, switches are superior to hubs. Hubs mirror all traffic to all computers, whereas switches are smart enough to send packets only to the intended destination. The result is that switches let two pairs of computers engage in full-speed data transfers with each other; with a hub, these two transfers would interfere with each other. Switches also allow *full-duplex* transmission, in which both parties can send data at the same time (like two people talking on a telephone). Hubs permit only *half-duplex* transmission, in which the two computers must take turns (like two people using walkie-talkies).



**NOTE**

A hub or switch is located centrally in a logical sense, but it doesn't have to be so located geographically. An approximately central location may help simplify wiring, but when you decide where to put the device, you should take into account the layout of your computers, your rooms, and available conduits between rooms.

Wireless networks typically rely on a *wireless access point (WAP)*, which acts something like a hub or switch in a wired network. WAPs also bridge together wired and wireless networks, enabling wired and wireless computers to communicate with each other. A step beyond a WAP is a wireless router, which combines WAP, Ethernet switch, and broadband router functions. Wireless routers are extremely popular in homes since they enable wired and wireless computers to share broadband Internet access, even when the Internet service provider (ISP) provides only one network address.

## Network Packets

Modern networks operate on discrete chunks of data known as *packets*. Suppose you want to send a 100KB file from one computer to another. Rather than send the file in one burst of data, you break it down into smaller chunks. You might send 100 packets of 1KB each, for

instance. This way, if there's an error sending one packet, you can resend just that one packet rather than the entire file. (Many network protocols include error-detection capabilities.)

Typically, each packet includes an *envelope* (which includes the sender address, the recipient address, and other housekeeping information) and a *payload* (which is the data intended for transmission). When the recipient system receives packets, it must hold onto them and reassemble them in the correct order to re-create the complete data stream. It's not uncommon for packets to be delayed or even lost in transmission, so error-recovery procedures are critical for protocols that handle large transfers. Some types of error recovery are handled transparently by the networking hardware, but others are done in software.

There are several types of packets, and they can be stored within each other. For instance, Ethernet includes its own packet type (known as a *frame*), and the packets generated by networking protocols that run atop Ethernet, such as those described in the next section, "Network Protocol Stacks," are stored within Ethernet frames. All told, a data transfer can involve several layers of wrapping and unwrapping data. With each layer, packets from the layer above may be merged or split up.

## Network Protocol Stacks

The packing and unpacking of network data is frequently described in terms of a *protocol stack*. Knowing how the pieces of such a stack fit together can help you understand networking as a whole, including the various network protocols used by Linux. Therefore, this section presents this information; it starts with a description of protocol stacks in general and moves on to the TCP/IP stack and alternatives to it.

### What Is a Protocol Stack?

It's possible to think of network data at various levels of abstractness. For instance, at one level, a network carries data packets for a specific network type (such as Ethernet) that are addressed to specific computers on a local network. Such a description, while useful for understanding a local network, isn't very useful for understanding higher-level network protocols, such as those that handle e-mail transfers. These high-level protocols are typically described in terms of commands sent back and forth between computers, frequently without reference to packets. The addresses used at different levels also vary, as explained in the upcoming section "Types of Network Addresses."

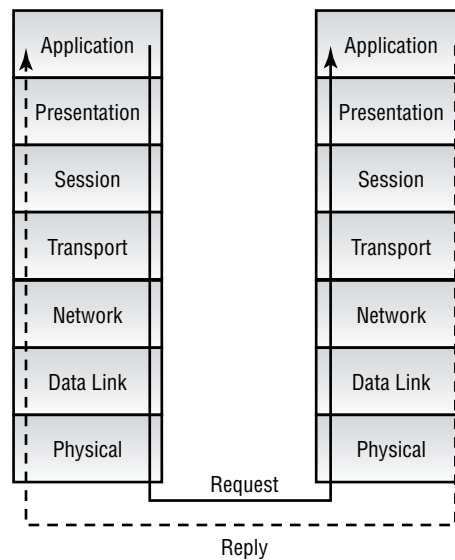
A protocol stack is a set of software that converts and encapsulates data between layers of abstraction. For instance, the stack can take the commands of e-mail transfer protocols, and the e-mail messages that are transferred, and package them into packets. Another layer of the stack can take these packets and repackage them into Ethernet frames. There are several layers to any protocol stack, and they interact in highly specified ways. It's often possible to swap out one component for another at any given layer. For instance, at the top of each stack is a program that uses the stack, such as an e-mail client. You can switch from one e-mail client to another without too much difficulty; both rest atop the same stack. Likewise, if you change a network card, you have to change the driver for that card, which constitutes a layer very low in the stack. Applications above that driver can remain the same.

Each computer in a transaction requires a compatible protocol stack. When they communicate, the computers pass data down their respective stacks and then send data to the partner system, which passes the data up its stack. Each layer on the receiving system sees the data as packaged by its counterpart on the sending computer.

## The OSI Model

The interactions of a protocol stack should become clearer with an example. A common model used for describing protocol stacks generically is the *Open System Interconnection (OSI) model*, illustrated in Figure 8.2. This model breaks networking tasks down into seven layers, from the Application layer (in which users' clients and the servers to which they connect run) to the Physical layer (which consists of network hardware like Ethernet cards). Each layer in between these does some task related to the packaging of data for transport or its unpacking.

**FIGURE 8.2** Information travels “down” and “up” protocol stacks, being checked and packed at each step of the way.



Each component layer of the sending system is equivalent to a layer on the receiving system, but these layers need not be absolutely identical. For instance, you can have different models of network card at the Physical layer, or you can even use entirely different network hardware types, such as Ethernet and Token Ring, if some intervening system translates between them. The computers may run different OSs entirely and hence use different—but logically equivalent—protocol stacks. What's important is that the stacks operate in compatible ways.



## The TCP/IP Protocol Stack

The OSI model describes an idealized protocol stack, although literal implementations of it do exist. One of the most common real-world network stacks is the *Transmission Control Protocol/Internet Protocol (TCP/IP)* stack. This stack is generally described using just four layers (Application, Transport, Internet, and Link), as opposed to OSI's seven (shown in Figure 8.2). The principles are the same for both models; the differences are just a matter of how the terms are applied and precisely how the stacks are implemented.

TCP/IP has several important features that make it a popular network protocol and the one on which the Internet is based. These characteristics include the following:

**Routable** TCP/IP was designed so that computers configured in a particular manner could route packets between two networks. These computers (known as *gateways* or *routers*) make the Internet possible. A small network links to another one via a router, which links to another, and so on. Such a collection of networks is known as an *internet* (without capitalization). The *Internet* (capitalized) is a particularly large globe-spanning internet.

**Flexible addressing system** TCP/IP supports two types of addresses, one based on numbers and one based on text. The current most popular numeric system supports approximately 4 billion addresses, and the textual system supports multiple levels of names. Both features support large and complex network structures.

**Multiple connection types** TCP/IP supports several types of connection, including the Transmission Control Protocol (TCP) after which the stack is named, the User Datagram Protocol (UDP), and the Internet Control Message Protocol (ICMP). These connection protocols support differing levels of complexity and error correction.

**Standards-based** The TCP/IP stack and many of the protocols that use it are described by documents maintained by the Internet Engineering Task Force (IETF; <http://www.ietf.org>), an international standards organization. IETF protocols are nonproprietary, so they may be implemented by anybody who cares to examine and put to use the IETF standards documents, which are known as *Requests for Comments (RFCs)*.

This combination has made TCP/IP a formidable protocol stack. It has been implemented in a large array of OSs, ranging from DOS to Linux. A huge number of network tools are built atop TCP/IP, including everything related to the Internet—Web browsers, e-mail clients, and so on. A few networking programs, though, either don't use TCP/IP or use it only optionally. Other protocol stacks remain popular in certain environments, and you should be aware of them and how they interact and compete with TCP/IP.

## Alternatives to TCP/IP

TCP/IP was initially developed using Unix, but today it is supported by many other platforms. Some of these other OSs have their own protocol stacks. Most of these have also been implemented on other OSs, including Linux. These TCP/IP alternatives don't support as many networking applications, though, and they're generally limited to use on much



smaller networks than TCP/IP supports. Nonetheless, some sites still run these stacks, which include the following:

**NetBEUI** IBM and Microsoft have been the driving forces behind *NetBEUI*, which is a nonroutable protocol stack that was developed for local networks of DOS and, later, OS/2 and Windows systems. NetBEUI is closely associated with *NetBIOS*, on which Microsoft's file-sharing protocols are built. For this reason, many Windows networks make extensive use of NetBEUI. It's also possible to use NetBIOS over TCP/IP, and this is the approach that Linux's Samba file server package uses to interconnect with Windows clients. Linux doesn't include a NetBEUI stack of its own, although an old add-on stack for 2.4.x kernels is available; see <http://zhubr.tamb.ru> for details. Chances are you won't need to use this stack because Samba works well over TCP/IP, and Samba is the only Linux package that might use a NetBEUI stack.

**IPX/SPX** The *Internet Packet Exchange (IPX)* and *Sequenced Packet Exchange (SPX)* protocols constitute a protocol stack that's similar in broad strokes to TCP/IP or NetBEUI. IPX/SPX was the core of Novell's networking tools through NetWare 5.0, although later versions use TCP/IP by default. Novell's networking software competes for file and printer sharing in DOS and Windows networks against NetBEUI and its associated tools. IPX/SPX support is included in the Linux kernel, although it might not be compiled by default in all kernels. File- and printer-sharing packages are also available that use the IPX/SPX stack. IPX/SPX are routable, but they aren't as amenable to creation of globe-spanning internet-networks as is TCP/IP.

**AppleTalk** Apple developed the *AppleTalk* stack for use with its LocalTalk network hardware. Its main use is with the AppleShare file-sharing protocols. Although initially tied to LocalTalk, AppleTalk can now be used over Ethernet (a combination that's sometimes called EtherTalk). The Linux kernel includes support for AppleTalk, but this may not be compiled in all kernels. The Linux package that supports AppleTalk and AppleShare is Netatalk (<http://netatalk.sourceforge.net>). Netatalk supports not just the old AppleTalk, but AppleShare IP, which uses TCP/IP as the protocol stack for file sharing. Mac OS X doesn't rely on AppleTalk nearly as much as its predecessors did; thus, AppleTalk is less important for modern Macintosh-dominated networks than it once was. You're most likely to need this support if you have very old (pre-OS X) versions of Mac OS or equally old networked Apple printers.

These alternatives to TCP/IP are all used on local networks, not on the Internet at large, which is a TCP/IP-based network. All of these alternatives are limited in ways that restrict their expansion. For instance, they lack the capacity to handle more than a couple of levels in their machine names. That is, as described in the upcoming section "Hostnames," TCP/IP supports a hierarchical name structure that reduces the chance of conflicts in names, enabling every computer connected to the Internet to have a unique name. The naming schemes of these alternative stacks are much simpler, making it extremely impractical to maintain a worldwide naming system.

Different protocol stacks are incompatible in the sense that they aren't completely interchangeable—for instance, you can't run an FTP client using AppleTalk. (A few protocols, like those used for Windows file sharing, can bind to multiple protocol stacks, though.) In another sense, these protocol stacks are *not* incompatible. Specifically, you can run multiple protocol stacks on one network or one computer. Many local networks today run two, three, or more protocol stacks. For instance, an office with both Macintoshes and Windows systems might run TCP/IP, NetBEUI, and AppleTalk.



### Real World Scenario

#### The Coming of IPv6

Another alternative protocol stack is actually an extension of TCP/IP. In 2009, the most popular version of the IP portion of TCP/IP is 4. A major upgrade to this is in the works, however, and it goes by the name *IPv6*, for IP version 6. IPv6 adds several features and improvements to TCP/IP, including standard support for more secure connections and support for many more addresses. Check <http://www.ipv6.org> for detailed information on IPv6.

Although the 4 billion addresses allowed by TCP/IP sounds like plenty, those addresses have not been allocated as efficiently as possible. Therefore, as the Internet has expanded, the number of truly available addresses has been shrinking at a rapid rate. IPv6 raises the number of addresses to  $2^{128}$ , or  $3.4 \times 10^{38}$ . This is enough to give every square millimeter of land surface on Earth  $2.2 \times 10^{18}$  addresses.

IPv6 is starting to emerge as a real networking force in many parts of the world. The United States, though, is lagging behind on IPv6 deployment. The Linux kernel includes IPv6 support, so you can use it if you need to do so. Chances are that by the time the average office will need IPv6, it will be standard. Configuring a system for IPv6 is somewhat different from configuring it for IPv4, which is what this chapter describes and what the Linux+ exam requires you to understand.

## Network Addressing

In order for one computer to communicate with another over a network, the computers need to have some way to refer to each other. The basic mechanism for doing this is provided by a network address, which can take several different forms, depending on the type of network hardware, protocol stack, and so on. Large and routed networks pose additional challenges to network addressing, and TCP/IP provides answers to these challenges. Finally, to address a specific program on a remote computer, TCP/IP uses a *port number*, which identifies a specific running program, something like the way a telephone extension number identifies an individual in a large company. This section describes all these methods of addressing.

## Types of Network Addresses

Consider an Ethernet network. When an Ethernet frame leaves one computer, it is normally addressed to another Ethernet card. This addressing is done using low-level Ethernet features, independent of the protocol stack in question. Recall, however, that the Internet is composed of many different networks that use many different low-level hardware components. A user might have a dial-up telephone connection (through a serial port) but connect to one server that uses Ethernet and another that uses Token Ring. Each of these devices uses a different type of low-level network address. TCP/IP requires something more to integrate across different types of network hardware. In total, three types of addresses are important when you are trying to understand network addressing: network hardware addresses, numeric IP addresses, and text-based hostnames.

### Hardware Addresses

At the lowest level of the OSI model is the Physical layer, which corresponds to network hardware. One of the characteristics of dedicated network hardware such as Ethernet or Token Ring cards is that they have unique *hardware addresses*, also known as *Media Access Control (MAC) addresses*, programmed into them. In the case of Ethernet, these addresses are 6 bytes in length, and they're generally expressed as hexadecimal (base 16) numbers separated by colons. You can discover the hardware address for an Ethernet card by using the `ifconfig` command. Type **`ifconfig ethn`**, where *n* is the number of the interface (0 for the first card, 1 for the second, and so on). You'll see several lines of output, including one like the following:

```
eth0 Link encap:Ethernet HWaddr 00:A0:CC:24:BA:02
```

This line tells you that the device is an Ethernet card and that its hardware address is 00:A0:CC:24:BA:02. What use is this, though? Certain low-level network utilities and hardware use the hardware address. For instance, network switches use it to direct data packets. The switch learns that a particular address is connected to a particular wire, and so it sends data directed at that address *only* over the associated wire. The *Dynamic Host Configuration Protocol (DHCP)*, which is described in the upcoming section “DHCP Configuration,” is a means of automating the configuration of specific computers. It has an option that uses the hardware address to consistently assign the same IP address to a given computer. In addition, advanced network diagnostic tools are available that let you examine packets that come from or are directed to specific hardware addresses.

For the most part, though, you don't need to be aware of a computer's hardware address. You don't enter it in most utilities or programs. It's important for what it does in general.

### IP Addresses

Earlier, I said that TCP/IP, at least in its IPv4 incarnation, supports about 4 billion addresses. This figure is based on the size of the *IP address* used in TCP/IP: 4 bytes (32 bits). Specifically,  $2^{32} = 4,294,967,296$ . For IPv6, 16-byte (128-bit) addresses are used. Not all of these addresses are usable; some are overhead associated with network definitions, and some are reserved.

The 4-byte IPv4 (or 16-byte IPv6) address and 6-byte Ethernet address are mathematically unrelated. Instead, the TCP/IP stack converts between the two using the *Address Resolution Protocol (ARP)* for IPv4 or the *Neighbor Discovery Protocol (NDP)* for IPv6. These protocols enable a computer to send a *broadcast* query—a message that goes out to all the computers on the local network. This query asks the computer with a given IP address to identify itself. When a reply comes in, it includes the hardware address, so the TCP/IP stack can direct traffic for a given IP address to the target computer's hardware address.



The procedure for computers that aren't on the local network is more complex. For such computers, a router must be involved. Local computers send packets destined to distant addresses to routers, which send the packets on to other routers or to their destination systems.

IPv4 addresses are usually expressed as four base-10 numbers (0–255) separated by periods, as in 192.168.29.39. If your Linux system's protocol stack is already up and running, you can discover its IPv4 address by using `ifconfig`, as described earlier. The output includes a line like the following, which identifies the IP address (`inet addr`):

```
inet addr:192.168.29.39 Bcast:192.168.29.255 Mask:255.255.255.0
```

Although it isn't obvious from the IP address alone, this address is broken down into two components: a network address and a computer address. The network address identifies a block of IP addresses that are used by one physical network, and the computer address identifies one computer within that network. The reason for this breakdown is to make the job of routers easier—rather than record how to direct packets destined for each of the 4 billion IP addresses, routers can be programmed to direct traffic based on packets' network addresses, which is a much simpler job.

IPv6 addresses work in a similar way, except that they're larger. Specifically, IPv6 addresses consist of eight groups of four-digit hexadecimal numbers separated by colons, as in `fed1:0db8:85a3:08d3:1319:8a2e:0370:7334`. If one or more groups of four digits is 0000, that group or those groups may be omitted, leaving two colons. Only one such group of zeroes may be compressed in this way, because if you removed two groups, there would be no way of telling how many sets of zeroes would have to be replaced in each group.

The *network mask* (also known as the *subnet mask* or *netmask*) is a number that identifies the portion of the IP address that's a network address and the part that's a computer address. It's helpful to think of this in binary (base 2) because the netmask uses binary 1 values to represent the network portion of an address and binary 0 values to represent the computer address. The network portion ordinarily leads the computer portion. Expressed in base 10, these addresses usually consist of 255 or 0 values, 255 being a network byte and 0 being a computer byte. If a byte is part network and part computer address, it will have some other value. Another way of expressing a netmask is as a single number representing the number of network bits in the address. This number usually follows the IP address and a slash. For instance, 192.168.29.39/24 is equivalent to 192.168.29.39 with a netmask of 255.255.255.0—the last number shows the network portion to be three solid 8-bit bytes,

in other words, 24 bits. The longer notation showing all four bytes of the network mask is referred to as *dotted quad* notation. IPv6 netmasks work just like IPv4 netmasks, except that larger numbers are involved, and IPv6 favors hexadecimal to decimal notation.

IP addresses and netmasks are extremely important for network configuration. If your network doesn't use DHCP or a similar protocol to assign IP addresses automatically, you must configure your system's IP address manually. A mistake in this configuration can cause a complete failure of networking or more subtle errors, such as an inability to communicate with just some computers.



Non-TCP/IP stacks have their own addressing methods. NetBEUI uses machine names; it has no separate numeric addressing method. AppleTalk uses two 16-bit numbers. These addressing schemes are independent from IP addresses.

Traditionally, IPv4 addresses have been broken into one of several classes, as shown in Table 8.1. These classes vary in size, the intention being to assign address ranges in particular classes to organizations based on an organization's size. This arrangement was intended to simplify the task of routers. In practice, though, IPv4 network classes are often assigned using *Classless Inter-Domain Routing (CIDR)*, in which any address range may be assigned in an arbitrary way—for instance, networks with only 255 addresses (that is, sized as Class C addresses) within the Class A address range. This distinction is important mainly because some tools for assigning IP addresses set defaults based on the apparent class of an address. You may need to change the default if your network doesn't conform to the usual class assignments. Classes A, B, and C are used for ordinary computers. Class D is intended for multicasting (sending the same data to multiple computers), while Class E is reserved.

**TABLE 8.1** IPv4 Network Classes

| Class | Address Range             | Network Mask  | Hosts per Network | Reserved Private Addresses  |
|-------|---------------------------|---------------|-------------------|-----------------------------|
| A     | 1.0.0.0–127.255.255.255   | 255.0.0.0     | 16,777,214        | 10.0.0.0–10.255.255.255     |
| B     | 128.0.0.0–191.255.255.255 | 255.255.0.0   | 65,534            | 172.16.0.0–172.31.255.255   |
| C     | 192.0.0.0–223.255.255.255 | 255.255.255.0 | 254               | 192.168.0.0–192.168.255.255 |
| D     | 224.0.0.0–239.255.255.255 | Not defined   | Not defined       | None                        |
| E     | 240.0.0.0–255.255.255.255 | Not defined   | Not defined       | None                        |

A few addresses have special meanings. The reserved private addresses (also referred to as RFC1918 addresses, after the standards document in which they're defined) are not used on the Internet at large, and most routers drop packets sent to these addresses. The intention is that these addresses may be used on local networks, for systems that don't connect to the Internet or that connect only via a network address translation (NAT) router, which "hides" client systems using its own address. This is useful if you have more systems than IP addresses. It also has security benefits, since it's impossible for an outside system to initiate a connection with an internal computer unless you explicitly configure the NAT router to handle this task.

Addresses in the 127.0.0.0/8 netblock refer to the local computer. Typically, 127.0.0.1 is used for generic connections to the computer itself. A few protocols can use additional addresses within that netblock for special purposes, but such uses are rare.

IPv6 has its equivalent to reserved private addresses. IPv6 site-local addresses may be routed within a site but not off-site. They begin with the hexadecimal number fec, fed, fee, or fef. Link-local addresses are restricted to a single network segment; they shouldn't be routed at all. These addresses begin with the hexadecimal number fe8, fe9, fea, or feb.

## Hostnames

Computers work with numbers, so it's not surprising that TCP/IP uses numbers as computer addresses. People, though, work better with names. For this reason, TCP/IP includes a way to link names for computers (known as *hostnames*) to IP addresses. In fact, there are *several* ways to do this, some of which are described in the next section, "Resolving Hostnames."

As with IP addresses, hostnames are composed of two parts: *machine names* and *domain names*. The former refers to a specific computer and the latter to a collection of computers. Domain names are not equivalent to the network portion of an IP address, though; they're completely independent concepts. Domain names are registered for use by an individual or organization, which may assign machine names within the domain and link those machine names to any arbitrary IP address desired. Nonetheless, there is frequently some correspondence between domains and network addresses because an individual or organization that controls a domain is also likely to want a block of IP addresses for the computers in that domain.

Internet domains are structured hierarchically. At the top of the hierarchy are the top-level domains (TLDs), such as .com, .edu, and .uk. These TLD names appear at the *end* of an Internet address. Some correspond to nations (such as .uk and .us, for the United Kingdom and for the United States, respectively), but others correspond to particular types of entities (such as .com and .edu, which stand for commercial and educational organizations, respectively). Within each TLD are various domains that identify specific organizations, such as sybex.com for Sybex or loc.gov for the Library of Congress. These organizations may optionally break their domains into *subdomains*, such as cis.upenn.edu for the Computer and Information Science department at the University of Pennsylvania. Even subdomains may be further subdivided into their own subdomains; this structure can continue for many levels but usually doesn't. Domains and subdomains include specific computers, such as www.sybex.com, Sybex's Web server.

When you configure your Linux computer, you may need to know its hostname. This will be assigned by your network administrator and will be within your organization's domain. If your computer isn't part of an organizational network (say, if it's a system that doesn't connect to the Internet at all or if it connects only via a dial-up account), you'll have to make up a hostname. Alternatively, you can register a domain name, even if you don't use it for running your own servers. Check <http://www.icann.org/en/registrars/accredited-list.html> for pointers to accredited domain registrars. Most registrars charge between \$10 and \$15 per year for domain registration. If your network uses DHCP, it may or may not assign your system a hostname automatically.



If you make up a hostname, choose an invalid TLD, such as `.invalid`. This will guarantee that you don't accidentally give your computer a name that legitimately belongs to somebody else. Such a name conflict could prevent you from contacting that system, and it could cause other problems as well, such as misdirected e-mail.

## Resolving Hostnames

The *Domain Name System (DNS)* is a distributed database of computers that convert between IP addresses and hostnames. Every domain must maintain at least two DNS servers that can either provide the names for every computer within the domain or redirect a DNS query to another DNS server that can better handle the request. Therefore, looking up a hostname involves querying a series of DNS servers, each of which redirects the search until the server that's responsible for the hostname is found. In practice, this process is hidden from you because most organizations maintain DNS servers that do all the dirty work of chatting with other DNS servers. You need only point your computer to your organization's DNS servers. This detail may be handled through DHCP, or it may be information you need to configure manually, as described later in the section "Basic Network Configuration."

Sometimes, you need to look up DNS information manually. You might do this if you know the IP address of a server through non-DNS means and suspect your DNS configuration is delivering the wrong address, or you might want to check whether a DNS server is working at all. Several programs can be helpful in performing such checks:

**nslookup** This program performs DNS lookups (on individual computers, by default) and returns the results. It also sports an interactive mode in which you can perform a series of queries. This program is officially deprecated, meaning that it's no longer being maintained and will eventually be dropped from its parent package (`bind-utils` or `bind-tools` on most distributions). Thus, you should get in the habit of using `host` or `dig` instead of `nslookup`.

**host** This program serves as a replacement for the simpler uses of `nslookup`, but it lacks an interactive mode, and of course many details of its operation differ. In the simplest case, you can type `host target.name`, where `target.name` is the hostname or IP address you want



to look up. You can add various options that tweak its basic operation; consult the `host` man page for details.

**dig** This program performs more complex DNS lookups than `host`. It can be used to obtain information on several critical systems within an entire domain. For instance, passing `MX` after the domain name produces output that includes the identities of the domain's mail server computers. Although you can use it to find the IP address for a single hostname (or a hostname for a single IP address), it's more flexible—but also more complex—than `host`.

Sometimes DNS is overkill. For instance, you might just need to resolve a handful of hostnames. This might be because you're configuring a small private network that's not connected to the Internet at large or because you want to set up a few names for local (or even remote) computers that aren't in the global DNS database. For such situations, `/etc/hosts` may be just what you need. This file holds mappings of IP addresses to hostnames, on a one-line-per-mapping basis. Each mapping includes at least one name, and sometimes more:

```
127.0.0.1 localhost
192.168.7.23 apollo.luna.edu apollo
```

In this example, the name `localhost` is associated with the `127.0.0.1` address, and the names `apollo.luna.edu` and `apollo` are tied to `192.168.7.23`. The first of these linkages is standard; it should exist in any `/etc/hosts` file. The second linkage is an example that you can modify as you see fit. The first name is a full hostname, including the domain portion; subsequent names on the line are aliases—typically the hostname without its full domain specification.

Once you've set up an `/etc/hosts` file, you can refer to computers listed in the file by name, whether or not those names are recognized by the DNS servers the computer uses. One major drawback to `/etc/hosts` is that it's a purely local file; setting a mapping in one computer's `/etc/hosts` file affects only those name lookups performed by that computer. Thus, to be an effective tool on an entire network, the `/etc/hosts` files must be modified on all of the computers on the network.

Linux normally performs lookups in `/etc/hosts` before it uses DNS. You can, however, modify this behavior by editing the `hosts` line in the `/etc/nsswitch.conf` file, which lists the order of the files and `dns` options, which stand for `/etc/hosts` and DNS, respectively. Very old programs that use old versions of the C library (`libc4` or `libc5`) rather than the newer `glibc` look to the `/etc/host.conf` file and its `order` line instead of `nsswitch.conf`. Change the order of the `hosts` and `bind` items in this file to match the order of the files and `dns` items in `/etc/nsswitch.conf`.

## Network Ports

Contacting a specific computer is important, but one additional type of addressing is still left: the sender must have an address for a specific program on the remote system. For instance, suppose you're using a Web browser. The Web server computer may be running more servers than just a Web server—it might also be running an e-mail server or an FTP



server, to name just two of many possibilities. Another number beyond the IP address enables you to direct traffic to a specific program. This number is a network port number, and every program that accesses a TCP/IP network does so through one or more ports.

When they start up, servers tie themselves to specific ports, which by convention are associated with specific server programs. For instance, port 25 is associated with e-mail servers, and port 80 is used by Web servers. Thus, a client can direct its request to a specific port and expect to contact an appropriate server. The client's own port number isn't fixed; it's assigned by the OS. Because the client initiates a transfer, it can include its own port number in the connection request, so clients don't need fixed port numbers. Assigning client port numbers dynamically also enables one computer to easily run several instances of a single client because they won't compete for access to a single port.

Fortunately, for basic functioning, you need to do nothing to configure ports on a Linux system. You may have to deal with this issue if you run unusual servers, though, because you may need to configure the system to link the servers to the correct ports.

### Clients and Servers

One important distinction is the one between clients and servers. A *client* is a program that initiates a network connection to exchange data. A server listens for such connections and responds to them. For instance, a Web browser, such as Mozilla Firefox or Opera, is a client program. You launch the program and direct it to a Web page, which means that the Web browser sends a request to the Web server at the specified address. The Web server sends back data in reply to the request. Clients can also send data, however, as when you enter information in a Web form and click a Submit or Send button.

The terms “client” and “server” can also be applied to entire computers that operate mostly in one or the other role. Thus, a phrase such as “Web server” is somewhat ambiguous—it can refer either to the Web server program or to the computer that runs that program. When this distinction is important and unclear from context, I clarify it (for instance, by referring to “the Web server program”).

## Basic Network Configuration

Now that you know something about how networking functions, a question arises: how do you implement networking in Linux? Most Linux distributions provide you with the means to configure a network connection during system installation, as mentioned in Chapter 1, “Getting Started with Linux.” Therefore, chances are good that networking already functions on your system. In case it doesn't, though, this section summarizes what you must

do to get the job done. Actual configuration can be done using either the automatic DHCP tool or static IP addresses. Linux's underlying network configuration mechanisms rely on startup scripts and their configuration files, but you may be able to use GUI tools to do the job instead.

## Network Hardware Configuration

The most fundamental part of network configuration is getting the network hardware up and running. In most cases, this task is fairly automatic—most distributions ship with system startup scripts that autodetect the network card and load the correct driver module. If you recompile your kernel, building the correct driver into the main kernel file will also ensure that it's loaded at system startup.

If your network hardware isn't correctly detected, though, subsequent configuration (as described in the upcoming sections “DHCP Configuration” and “Static IP Address Configuration”) won't work. To correct this problem, you must load your network hardware driver. You can do this with the `modprobe` command:

```
modprobe tulip
```

You must know the name of your network hardware's kernel module, though (`tulip` in this example). This name may not be immediately obvious, because it varies greatly depending on your hardware, and the name is usually based on the chipset used in the network card, rather than on the name of the network card or its manufacturer. Chapter 1 includes information on identifying hardware, so consult it if you have problems with this task.

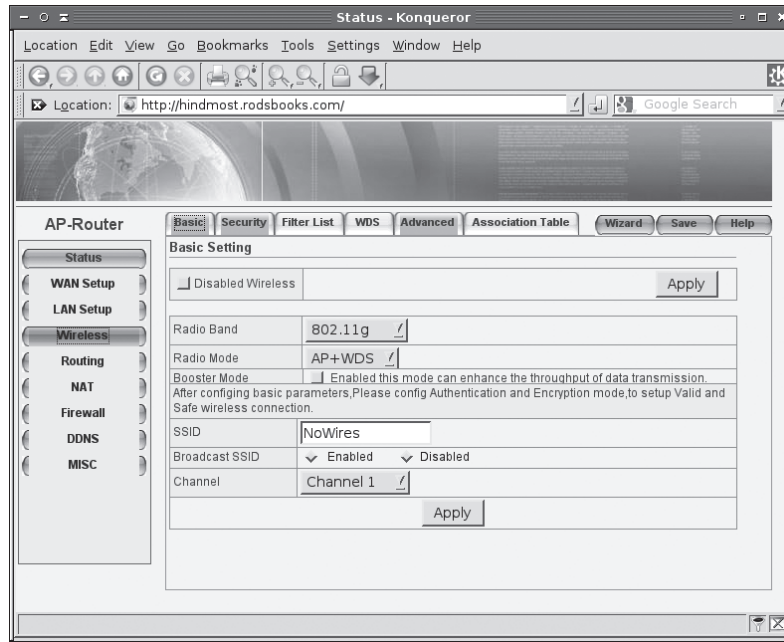
Once Linux has recognized the network hardware, you should be able to continue with network configuration, as described in the next couple of sections. To make Linux recognize your hardware at every boot, though, you may need to add the `modprobe` command to a startup script. This task can be tricky; most distributions use very convoluted startup scripts. These scripts *should* already be detecting and loading the network driver. If they don't, the best approach may be to recompile the kernel and build the driver into the main kernel file. Alternatively, you can try adding a call to `modprobe` to an appropriate network startup script; however, placing this call in a good location can be a difficult task. Chapter 4, “Managing System Services,” includes information on system startup scripts.

## Setting Wireless Options

If you're using a wireless (Wi-Fi) device, you must use the `iwconfig` command to set wireless-specific options before you use DHCP or manual tools to bring up the network interface. This is necessary because wireless networks have their own unique features, such as network names, broadcast frequencies, and encryption keys, that are not applicable to wired networks. As with other network configuration tools, your distribution's startup scripts may launch `iwconfig` as part of a general network startup routine, so you may not need to do this manually; however, at the very least you'll need to enter the relevant information in a GUI or text-based tool, perhaps when you install Linux.

To begin the process, you must first obtain information on your wireless network's settings. If you're using a wireless network set up by somebody else, such as your employer's network or a public access point, you should be able to get the relevant information from its maintainer. If you're using a network that you maintain yourself, such as a home network or one that you've set up for your employer, you can find the information from the configuration tools provided by your WAP or broadband router. This information can often be accessed via a Web server run on the device. For instance, Figure 8.3 shows the configuration screen for one broadband router. In this model, the Wireless options screen contains several tabs, each of which provides certain data. Pay particular attention to the radio band (802.11g in Figure 8.3), service set identifier (SSID; NoWires in Figure 8.3), channel (Channel 1 in Figure 8.3), authentication type, and authentication key. (These last two items are on the Security tab, which isn't shown in Figure 8.3.)

**FIGURE 8.3** WAPs and broadband routers provide configuration tools in which you set wireless options for your network.



If you're configuring your own wireless network, be sure to enable the highest security level possible, Wi-Fi Protected Access 2 (WPA2). Earlier security systems, and particularly Wired Equivalent Privacy (WEP), are extremely vulnerable. Using them enables anybody with modest skill to snoop on all your network traffic. Depending on your local security settings, intruders might be able to easily break into other wired or wireless computers on your network from outside your building using a notebook computer.

To use `iwconfig`, you pass it the relevant data using option names such as `essid` and `channel`, preceded by the wireless network device name (typically `wlan0`):

```
iwconfig wlan0 essid NoWires channel 1 mode Managed key s:N1mP7mHNw
```

This example sets the options for `wlan0` to use the managed network on channel 1 with the SSID of `NoWires` and a password of `N1mP7mHNw`. The password requires a few extra comments. Ordinarily, `iwconfig` takes a password as a string of hexadecimal values, with optional dashes between 2-byte blocks, as in `0123-4567-89AB`. Often, however, the password is given as a text string. The string `s:` must precede the password in this case, as shown in the example.

For details on additional `iwconfig` options, including some highly technical ones, consult its man page.

Once you've configured a wireless interface, you can check on its settings by using `iwconfig` with no options or with only the interface name:

```
iwconfig wlan0
wlan0 IEEE 802.11g ESSID:"NoWires"
 Mode:Managed Frequency:2.462 GHz Access Point: 08:10:74:24:1B:D4
 Bit Rate=54 Mb/s Tx-Power=27 dBm
 Retry min limit:7 RTS thr:off Fragment thr=2352 B
 Encryption key: 314E-506D-6d37-4E48-0A [2]
 Link Quality=100/100 Signal level=-32 dBm Noise level=-94 dBm
 Rx invalid nwid:0 Rx invalid crypt:0 Rx invalid frag:0
 Tx excessive retries:0 Invalid misc:0 Missed beacon:0
```

In addition to providing information on settings, `iwconfig` provides some diagnostic information, such as the link quality, received (Rx) and transmitted (Tx) errors, and so on.

## DHCP Configuration

One of the easiest ways to configure a computer to use a TCP/IP network is to use DHCP, which enables one computer on a network to manage the settings for many other computers. It works like this: when a computer running a DHCP client boots up, it sends a broadcast in search of a DHCP server. The server replies (using nothing but the client's hardware address) with the configuration information the client needs to allow it to communicate with other computers on the network—most importantly the client's IP address and netmask and the network's gateway and DNS server addresses. The DHCP server may also give the client a hostname. The client then configures itself with these parameters. The IP address is not assigned permanently; it's referred to as a *DHCP lease*, and if it's not renewed, the DHCP server may give the lease to another computer. Therefore, from time to time, the client checks back with the DHCP server to renew its lease.

Three DHCP clients are in common use on Linux: `pump`, `dhclient`, and `dhcpcd` (not to be confused with the DHCP server, `dhcpd`). Some Linux distributions ship with just one of

these, but others ship with two or even all three. All distributions have a default DHCP client, though—the one that’s installed when you tell the system you want to use DHCP at system installation time. Those that ship with multiple DHCP clients typically enable you to swap out one for another simply by removing the old package and installing the new one.

Ideally, the DHCP client runs at system bootup. This is usually handled either by a SysV startup file, as described in Chapter 4, or as part of the main network configuration startup file (typically a SysV startup file called `network` or `networking`). The system often uses a line in a configuration file to determine whether to run a DHCP client. For instance, Fedora Linux sets this option in a file called `/etc/sysconfig/network-scripts/ifcfg-eth0` (this filename may differ if you use something other than a single Ethernet interface). The line in question looks like this:

```
BOOTPROTO=dhcp
```

If the `BOOTPROTO` variable is set to something else, changing it as shown here will configure the system to use DHCP. It’s usually easier to use a GUI configuration tool to set this option, however, as described in the upcoming section “Using GUI Configuration Tools.”

In Ubuntu, the equivalent configuration is done in the `/etc/network/interfaces` file, which contains the following line for DHCP configuration on the first Ethernet device:

```
iface eth0 inet dhcp
```

Replacing `dhcp` with `static` in this line changes the system to use a static IP address, but you must then add lines to provide the static IP address and related information. As with Fedora, it may be easier to use a GUI configuration tool.

## Static IP Address Configuration

If a network lacks a DHCP server, you must provide basic network configuration options manually. You can set these options using interactive commands, as described shortly, but to set them in the long term, you adjust a configuration file. Details vary from one distribution to another, so you may have to search through your `/etc` directory to find the appropriate file to make permanent changes. By way of example, I show the files used in Fedora 10 and Ubuntu 8.04. DNS settings are done in the same way in all distributions. Temporary changes can be done using the same tools on any distribution, as well.



If you aren’t sure what to enter for the basic networking values (the IP address, network mask, gateway address, and DNS server addresses), you should consult your network administrator. *Do not* enter random values or values you make up that are similar to those used by other systems on your network. Doing so is unlikely to work at all, and it could conceivably cause a great deal of trouble—say, if you mistakenly use an IP address that’s reserved for another computer.

## Making Permanent Changes in Fedora

In Fedora, `/etc/sysconfig/network-scripts/ifcfg-eth0` controls most network settings. (The filename varies depending on the network interface.) Listing 8.1 shows a typical `ifcfg-eth0` file, configured to use a static IP address.

### Listing 8.1: A Sample Fedora Network Configuration File

```
DEVICE=eth0
BOOTPROTO=static
IPADDR=192.168.29.39
NETMASK=255.255.255.0
NETWORK=192.168.29.0
BROADCAST=192.168.29.255
GATEWAY=192.168.29.1
ONBOOT=yes
```

Several specific items are required, or at least helpful, for static IP address configuration:

**IP address** You can set the IP address manually via the `ifconfig` command (described in more detail shortly) or via the `IPADDR` item in the configuration file.

**Network mask** The netmask can be set manually via the `ifconfig` command or via the `NETMASK` item in a configuration file.

**Gateway address** You can manually set the gateway via the `route` command. To set it permanently, you need to adjust a configuration file, which may be the same configuration file that holds other options or another file, such as `/etc/sysconfig/network/routes`. In either case, the option is likely to be called `GATEWAY` or `gateway`. The gateway isn't necessary on a system that isn't connected to a wider network—that is, if the system works *only* on a local network that contains no routers.

The network configuration script may hold additional options, but most of these are related to others. For instance, Listing 8.1 has an option specifying the interface name (`DEVICE=eth0`), another that tells the computer to assign a static IP address (`BOOTPROTO=static`), and a third to bring up the interface when the computer boots (`ONBOOT=yes`). The `NETWORK` and `BROADCAST` items in Listing 8.1 are derived from the `IPADDR` and `NETMASK` items, but you can change them if you understand the consequences.

## Making Permanent Changes in Ubuntu

Ubuntu Linux uses the `/etc/network/interfaces` file to hold IP address, netmask, and gateway information for all its interfaces. Listing 8.2 presents an example of this file.

### Listing 8.2: A Sample Ubuntu Network Configuration File

```
The loopback network interface
auto lo
iface lo inet loopback
```

```
The primary network interface
auto eth0
iface eth0 inet static
address 192.168.29.39
netmask 255.255.255.0
gateway 192.168.29.1
```

The loopback interface configuration in the first few lines of the file should never be touched. The `eth0` configuration, though, can be altered to suit your needs. The entry includes the same basic information as appears in a Fedora configuration file, but the exact keywords and formatting differ. Also, pay attention to the line beginning with the keyword `iface`: it concludes with the word `static`. In a DHCP configuration, as noted earlier, it would end with `dhcp`, and the following `address`, `netmask`, and `gateway` lines would be absent.

## Setting DNS Options

In order for Linux to use DNS to translate between IP addresses and hostnames, you must specify at least one DNS server in the `/etc/resolv.conf` file. Precede the IP address of the DNS server by the keyword `nameserver`. You may include up to three such lines, although just one may be sufficient. (More than one provides redundancy in case the first name server becomes unresponsive.) This file may also include a `domain` keyword, which sets the Internet domain name in which the system resides, and one or more `search` keywords, which are additional domains that are searched if none is specified. For instance, to enable you to contact `dino.pangaea.edu` by using the shortened hostname `dino`, you could include the line `search pangaea.edu`, whether or not your computer is in the `pangaea.edu` domain. Listing 8.3 shows a sample `resolv.conf` file.

### Listing 8.3: A Sample `/etc/resolv.conf` File

```
domain pangaea.edu
search luna.edu
nameserver 192.168.29.11
nameserver 172.24.24.24
nameserver 10.103.9.41
```

Adjusting this file is all you need to do to set the name server addresses; you don't have to do anything else to make the setting permanent. You might notice that Listing 8.3 specifies name servers on three different networks. This is perfectly valid (although for example purposes I've used addresses on reserved private networks). In general, though, the closer a DNS server is to your own system, the better.



If you use DHCP, the DHCP client alters `/etc/resolv.conf`. Thus, if you make changes to this file and then use DHCP, your changes will be lost.

## Implementing Permanent Changes

If you've made manual changes to your configuration files, you can implement those changes by using the `ifup` and `ifdown` commands. These tools bring up or shut down a network interface, respectively, using the definitions in your configuration files. For instance, suppose you want to make changes to an existing configuration for `eth0`. Before making your changes, you would bring the interface down:

```
ifdown eth0
```

After making your changes, you would bring it up:

```
ifup eth0
```

The alternative to using these tools is to issue `ifconfig` and `route` commands to duplicate your changes, as described in the next section, “Setting Temporary Options.” Using `ifup` and `ifdown` is simpler and enables you to verify that your changes work the way you expect. Using the underlying `ifconfig` and `route` commands is better for making temporary changes, such as adding a notebook computer to a hotel network to which it will be connected only briefly.

## Setting Temporary Options

As mentioned earlier, the `ifconfig` program is critically important for setting both the IP address and netmask. This program can also display current settings. Basic use of `ifconfig` to bring up a network interface resembles the following:

```
ifconfig interface up addr netmask mask
```

For instance, the following command brings up `eth0` (the first Ethernet card) using the address 192.168.29.39 and the netmask 255.255.255.0:

```
ifconfig eth0 up 192.168.29.39 netmask 255.255.255.0
```

This command links the specified IP address to the card so that the computer will respond to the address and claim to be that address when sending data. It doesn't, though, set up a route for traffic beyond your current network. For that, you need to use the `route` command:

```
route add default gw 192.168.29.1
```

Substitute your own gateway address for 192.168.29.1. (Advanced routing and the `route` command are described in more detail in Chapter 9.) Both `ifconfig` and `route` can display information on the current network configuration. For `ifconfig`, omit `up` and everything that follows; for `route`, omit `add` and everything that follows. For instance, to view interface configuration, you might issue the following command:

```
ifconfig eth0
eth0 Link encap:Ethernet HWaddr 00:A0:CC:24:BA:02
 inet addr:192.168.29.39 Bcast:192.168.29.255 Mask:255.255.255.0
```



```
UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
RX packets:10469 errors:0 dropped:0 overruns:0 frame:0
TX packets:8557 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:100
RX bytes:1017326 (993.4 Kb) TX bytes:1084384 (1.0 Mb)
Interrupt:10 Base address:0xc800
```

When configured properly, `ifconfig` should show a hardware address (`HWaddr`), an IP address (`inet addr`), and additional statistics. There should be few or no errors, dropped packets, or overruns for both received (RX) and transmitted (TX) packets. Ideally, few (if any) collisions should occur, but some are unavoidable if your network uses a hub rather than a switch. If collisions total more than a few percent of the total transmitted and received packets, you may want to consider replacing a hub with a switch. To use `route` for diagnostic purposes, you might try the following:

```
route
Kernel IP routing table
Destination Gateway Genmask Flags Metric Ref Use Iface
192.168.29.0 * 255.255.255.0 U 0 0 0 eth0
127.0.0.0 * 255.0.0.0 U 0 0 0 lo
default 192.168.29.1 0.0.0.0 UG 0 0 0 eth0
```

This shows that data destined for 192.168.29.0 (that is, any computer with an IP address between 192.168.29.1 and 192.168.29.254) goes directly over `eth0`. The 127.0.0.0 network is a special interface that “loops back” to the originating computer. Linux uses this for some internal networking purposes. The last line shows the *default route*—everything that doesn’t match any other entry in the routing table. This line specifies the default route’s gateway system as 192.168.29.1. If it’s missing or misconfigured, some or all traffic destined for external networks, such as the Internet, won’t make it beyond your local network segment.

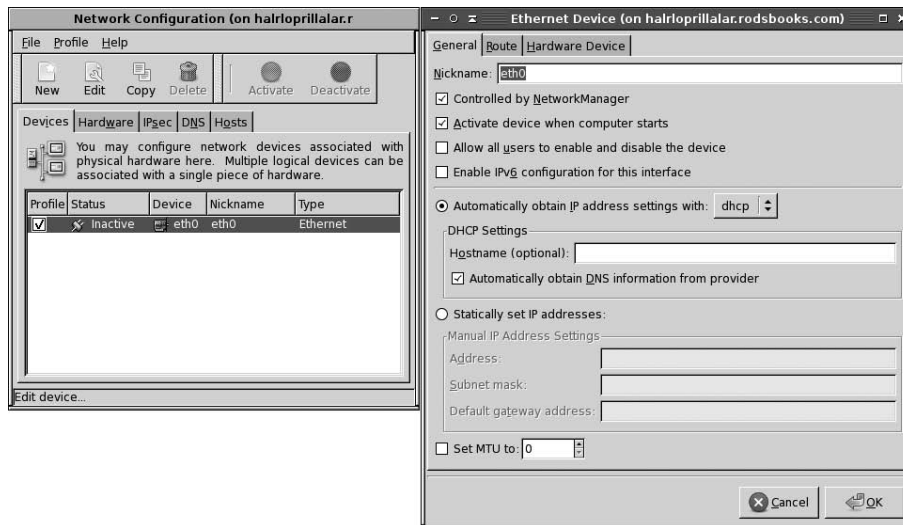
As with DHCP configuration, it’s almost always easier to use a GUI configuration tool to set up static IP addresses, at least for new administrators. The exact locations of the configuration files differ from one distribution to another, so the examples listed earlier may not apply to your system.

## Using GUI Configuration Tools

Most distributions include their own GUI configuration tools for network interfaces. For instance, Fedora and Red Hat ship with a custom GUI tool called Network Configuration (`system-config-network-gui`) and a text-mode tool called `system-config-network-tui`, SUSE has a text-mode and GUI tool called YaST, and Ubuntu ships with a GUI tool called Network Settings (`network-admin`). The details of operating these programs differ, but the GUI configuration tool provides a means to enter the information described earlier.

Figure 8.4 shows the Fedora Network Configuration tool, which you can use by typing **system-config-network-gui** or by selecting the System ➤ Administration ➤ Network menu item from the default GNOME desktop menu. Figure 8.4 shows both the main window (Network Configuration) and the one in which you can set the most basic settings for an individual device (Ethernet Device). To see the latter window, you must highlight a device (only one is available in Figure 8.4's Network Configuration window) and click Edit. You can then enter your static IP address or, as in Figure 8.4, click the Automatically Obtain IP Address Settings With button to use DHCP. (Alternatively, you can choose the older BootP protocol or configure a dial-up configuration with this tool.) Additional options, including those to set the route and DNS features, are available from other tabs on these two dialog boxes.

**FIGURE 8.4** GUI network configuration tools provide fields in which you enter basic networking parameters.



The precise details of how to configure a Linux system using GUI tools differ from one distribution to another. The basic principles are the same, though; you must choose whether to use static IP address assignment or an automatic system such as DHCP, and then you enter a number of key options, depending on what configuration method you choose.

If you use a wireless network, most desktop environments provide GUI tools to help you bring up a wireless link. Look for icons along your desktop environment's menu bars that might relate to wireless connections and click them. These tools vary greatly in the details of their operation, but most are easy to use and will help you make connections. You can usually save your settings to simplify moving a notebook computer between networks.



## Real World Scenario

### Using PPP

With the rise in broadband access, most Linux systems—even in homes—now have always-up Internet connections. Nonetheless, you might still need to use a dial-up PPP connection from time to time. The easiest way to do this generally is to employ a GUI configuration tool, such as those described in “Using GUI Configuration Tools.” These tools collect information such as the dial-in system’s telephone number, a username, and a password, and they alter the relevant configuration files to make initiating a PPP connection easy.

If you prefer to edit the configuration files manually, start with `/etc/ppp/pap-secrets` or `/etc/ppp/chap-secrets`. These files hold authentication information for dial-up connections:

```
username server password IP_address
```

You’ll normally enter a username and password, but place an asterisk (\*) in the *server* field and leave the *IP\_address* field blank.

With the authentication data entered, you can edit the `ppp-on` script, which is usually found somewhere in the `/usr/share/doc/ppp` directory tree. Copy the script, along with `ppp-off` and `ppp-on-dialer`, to a location such as `/usr/local`. You must then edit `ppp-on` to include your ISP’s telephone number (on the `TELEPHONE` line) and your username (on the `ACCOUNT` line). Verify that the `DIALER` line points to the `ppp-on-dialer` script. You may also need to modify the device used by the call to `pppd` at the end of the script (usually `/dev/ttyS0` by default); this device must point to your modem.

With these changes in place, you can test your connection by typing **ppp-on**. (Depending on security settings, you may need to be root to do this.) If it’s successful, your modem will dial and connect, and a short while later your PPP connection will come up. Typing **ifconfig ppp0** will show a valid network link, and you should be able to use ordinary Linux networking programs. Type **ppp-off** to shut down your PPP connection.

## Diagnosing Network Problems

Network configuration is a complex topic, and unfortunately, things don’t always work as planned. Fortunately, there are a few commands you can use to help diagnose a problem. Four of these are `arp`, `ping`, `tracert`, and `netstat`. Each of these commands exercises the network in a particular way and provides information that can help you track down the source of a problem. You can also use other network tools to help give your network connections a workout.

## Examining the ARP Cache

The `arp` utility examines or updates the ARP cache—the list of local IP addresses and their associated hardware addresses maintained by the computer. Used as a diagnostic tool, this utility can help you verify that some of the most fundamental parts of a network link are working.

Before using `arp` as a diagnostic tool, though, you must first at least attempt to contact a few other computers using other tools. Linux builds the ARP cache as network activity occurs. For instance, if you attempt to connect to 192.168.24.78, Linux sends a query to the network looking for this system, and, if that system replies, Linux adds that system to its ARP cache. Only after that point will the `arp` command show the association.

You should also keep in mind that `arp` is a *local* network tool; it returns data only on the computers that exist on the same network segment as the system on which it's run. That is, you won't get data on computers on the Internet at large; those systems are accessed via your network gateway system. You should see your Internet gateway system in the ARP cache, though—assuming you've accessed that gateway, either directly or by accessing systems to which it routes.

The simplest way to use `arp` is to type its name alone:

```
$ arp
Address HWtype HWaddress Flags Mask Iface
hindmost.rodsbooks.com ether 08:10:74:24:1b:d4 C eth0
halrloprillalar.rodsboo ether 00:0c:76:96:a3:73 C eth0
seeker.rodsbooks.com ether 00:e0:4c:ee:59:e6 C eth0
```

This use shows three systems in the ARP cache, identified by their hostnames, hardware types, hardware addresses, flags, masks, and network interfaces. If nothing appears in the ARP cache, then either you've made no network access attempts or something is seriously wrong with your configuration. Try using `ifconfig` to verify that the network interface is up, and check your network cabling. If several systems appear but one that should appear doesn't, perhaps that system is down, or perhaps the cabling between the two systems is at fault—for instance, a cable could be disconnected, or a switch might be faulty.

Several `arp` options enable you to limit the program's output. For instance, `-H type` limits output to specific hardware types, such as `ether` for Ethernet; `-i Iface` limits output to specific interfaces; and `-n` substitutes IP addresses for hostnames.

In addition to providing information, `arp` enables you to manipulate the ARP cache. This isn't normally necessary, but it could be helpful if you've reconfigured a computer or if you want to delete an entry that you believe is causing problems. The `-d address` option deletes the entry for a specified IP address, while `-s ip_address hw_address` adds an entry for the specified IP address and hardware address. These changes may not hold forever, though; the ARP table is dynamic, so any changes you make may be overridden as normal network operations occur.

## Testing Basic Connectivity

Another basic network test is the `ping` command, which sends a simple packet to the system you name (via IP address or hostname) and waits for a reply. In Linux, `ping` continues sending packets once every second or so until you interrupt it with a `Ctrl+C` keystroke. Alternatively, you can pass the `-c n` command-line option, which causes `ping` to send only *n* test packets. Here's an example of its output:

```
$ ping speaker
PING speaker.rodsbooks.com (192.168.1.1) from 192.168.1.3 : 56(84) bytes of data.
64 bytes from speaker.rodsbooks.com (192.168.1.1): icmp_seq=0 ttl=255 time=149 usec
64 bytes from speaker.rodsbooks.com (192.168.1.1): icmp_seq=1 ttl=255 time=136 usec
64 bytes from speaker.rodsbooks.com (192.168.1.1): icmp_seq=2 ttl=255 time=147 usec
64 bytes from speaker.rodsbooks.com (192.168.1.1): icmp_seq=3 ttl=255 time=128 usec

--- speaker.rodsbooks.com ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max/mdev = 0.128/0.140/0.149/0.008 ms
```

This command sent four packets and waited for their return, which occurred quite quickly (in an average of 0.140 milliseconds [ms]) because the target system was on the local network. By pinging systems on both local and remote networks, you can isolate where a network problem occurs. For instance, if you can ping local systems but not remote systems, the problem is most probably in your router configuration. If you can ping by IP address but not by name, the problem is with your DNS configuration.

Some computers are configured to ignore ping requests. If you attempt to ping such systems, you'll get no reply, which you might mistakenly believe means that the system is inaccessible when in fact it isn't. If you're in doubt, you can try other systems or try using a protocol to which you know the system should respond, as described shortly, in "Using General Network Tools."

## Tracing a Route

A step up from `ping` is the `traceroute` command, which sends a series of three test packets to each computer between your system and a specified target system. The result looks something like this:

```
$ traceroute -n 10.1.0.43
traceroute to 10.1.0.43 (10.1.0.43), 30 hops max, 52 byte packets
 1 192.168.1.254 1.021 ms 36.519 ms 0.971 ms
 2 10.10.88.1 17.250 ms 9.959 ms 9.637 ms
 3 10.9.8.173 8.799 ms 19.501 ms 10.884 ms
```

```

4 10.9.8.133 21.059 ms 9.231 ms 103.068 ms
5 10.9.14.9 8.554 ms 12.982 ms 10.029 ms
6 10.1.0.44 10.273 ms 9.987 ms 11.215 ms
7 10.1.0.43 16.360 ms * 8.102 ms

```

The `-n` option to this command tells `tracert` to display target computers' IP addresses, rather than their hostnames. This can speed up the process a bit, and it can sometimes make the output easier to read—but you might want to know the hostnames of problem systems, because that can help you pinpoint who's responsible for a problem.

This sample output shows a great deal of variability in response times. The first hop, to 192.168.1.254, is purely local; this router responded in 1.021, 36.519, and 0.971 ms to its three probes. (Presumably the second probe caught the system while it was busy with something else.) Probes of most subsequent systems are in the 8–20 ms range, although one is at 103.068 ms. The final system has only two times; the middle probe never returned, as the asterisk (\*) on this line indicates.

Using `tracert`, you can localize problems in network connectivity. Highly variable times and missing times can indicate a router that's overloaded or that has an unreliable link to the previous system on the list. If you see a dramatic jump in times, it typically means that the physical distance between two routers is great. This is common in intercontinental links. Such jumps don't necessarily signify a problem, though, unless the two systems are close enough that a huge jump isn't expected.

What can you do with the `tracert` output? Most immediately, `tracert` is helpful in determining whether a problem in network connectivity exists in a network for which you're responsible. For instance, the variability in the first hop of the preceding example could indicate a problem on the local network, but the lost packet associated with the final destination most likely is not a local problem. If the trouble link is within your jurisdiction, you can check the status of the problem system, nearby systems, and the network segment in general.



As with `ping`, some routers are configured to drop packets from `tracert`. Thus, the `tracert` output may indicate dropped packets even though regular network traffic passes through these routers just fine.

## Checking Network Status

Another useful diagnostic tool is `netstat`. This is something of a Swiss Army knife of network tools because it can be used in place of several others, depending on the parameters it is passed. It can also return information that's not easily obtained in other ways. Table 8.2 summarizes some examples.

TABLE 8.2 Common netstat Options

| Option       | Option Abbreviation | Meaning                                                                                                                                                                                                                                                                             |
|--------------|---------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| --interface  | -i                  | Pass netstat this parameter to obtain information on your network interfaces similar to what ifconfig returns. (Some versions of netstat return information in the same format, but others display the information differently.)                                                    |
| --route      | -r                  | You can use this parameter to obtain a routing table listing similar to what the route command displays.                                                                                                                                                                            |
| --masquerade | -M                  | Pass netstat this parameter to obtain information on connections mediated by Linux's NAT features, which often go by the name "IP masquerading." NAT enables a Linux router to "hide" a network behind a single IP address. This can be a good way to stretch limited IP addresses. |
| --program    | -p                  | Some versions of netstat support this parameter, which attempts to provide information on the programs that are using network connections. This attempt isn't always successful, but it often is, so you can see what programs are making outside connections.                      |
| Various      | Various             | When used with various other parameters, or without any parameters at all, netstat returns information on open ports and the systems to which they connect.                                                                                                                         |

Keep in mind that netstat is a very powerful tool, and its options and output aren't entirely consistent from one distribution to another. You may want to peruse its man page and experiment with it to learn what it can do.

## Name Server Troubleshooting

The nslookup, host, and dig tools were described earlier as a means to test basic name server functioning. One trick that's helpful if you encounter problems is to use another name server. You can do this by passing the name of a particular name server to these tools. With nslookup and host, you trail your query with the IP address of the server you

want to use; with `dig`, you begin the query with the name server's address preceded by an at-sign (@). For instance:

```
$ host www.oberlin.edu 192.168.1.254
$ dig @192.168.1.254 www.oberlin.edu
```

These queries both employ the DNS server at 192.168.1.254 instead of the computer's default DNS server. (I've omitted their outputs, which can be lengthy.) The result can help you identify servers that are down or perhaps server IP addresses you've entered incorrectly.



If `ifconfig` shows that your network interface is up but you can't seem to access the network, try using an IP address rather than a hostname. (You can use `host` on another computer to look up a hostname, or you can try your router's IP address.) If you can access a system by IP address but not by hostname, then the problem almost certainly lies with your DNS configuration—or perhaps your DNS server is down.

The `hostname` command can also be a useful diagnostic tool. This program returns the currently defined hostname for the computer:

```
$ hostname
aldrin.luna.edu
```

If your system's hostname is set incorrectly, you can run into some odd networking problems, including delays or an inability to reach systems on your local network via their short names. You can temporarily set a hostname by passing it to `hostname`, but if you need to set it permanently, you may need to do so in configuration files. Typically, editing `/etc/hostname` or `/etc/HOSTNAME` will do the job, or you can use a GUI configuration tool, such as the ones described earlier, in “Using GUI Configuration Tools.”



The hostname returned or set by the `hostname` command is unrelated to the hostname as maintained by your local network's DNS server. The two should match, but using `hostname` to set your hostname will *not* adjust the DNS entry and hence the way other computers contact yours. Setting your hostname locally, either via the `hostname` utility or by adjusting your configuration files, affects the hostname used in certain programs. For instance, your local hostname may affect your default e-mail address in mail clients, the local hostname used by mail servers, and the hostname displayed by Bash in some prompt configurations.

## Using General Network Tools

You can use any network client or server as a means of testing network connections. Sometimes this is helpful if a remote system is configured to ignore ping requests. For instance,



you might launch Mozilla Firefox to test the response of a computer that runs a Web server program, or you might use an e-mail client program to test a mail server computer.

One particularly helpful test program is `telnet`. This program is a simple text-mode remote login program, and when it's used for its original intended purpose, it enables you to connect to a remote system and run text-mode programs on it. The Telnet protocol, however, is unencrypted and so is no longer recommended for remote logins; instead, you should use Secure Shell (SSH) for this task. The `telnet` program remains useful as a network diagnostic tool, however, because you can pass a port number after the system name to connect to any TCP port on the target system:

```
$ telnet mail.example.com 25
Trying 192.168.1.2...
Connected to mail.
Escape character is '^]'.
220 mail.example.com ESMTP Postfix (Ubuntu)
```

This example shows a connection to port 25 on `mail.example.com`. This port is ordinarily used by the Simple Mail Transfer Protocol (SMTP), and the example shows an SMTP server (Postfix) responding on this port. In other words, the target system's mail server program is functioning, at least in a minimal way. If you can't send e-mail, therefore, the problem doesn't lie in basic connectivity.

If you know enough about the protocols involved, you can use `telnet` to simulate a data exchange session. For instance, the preceding exchange might be continued:

```
mail from:<ben@example.com>
250 2.1.0 Ok
rcpt to:<george@example.com>
250 2.1.5 Ok
data
354 End data with <CR><LF>.<CR><LF>
This is a message!
.
250 2.0.0 Ok: queued as 8B00C6735
quit
```

The `mail from:`, `rcpt to:`, `data`, and `quit` commands are all legal ones in SMTP, while the text of the message (`This is a message!`) is the body of an e-mail message. This example shows a message being accepted for delivery by the server, but if an error occurred, the server would present an error message that might be diagnostic—for instance, the server might complain that it's not configured to forward e-mail between sites.

Of course, to do more than check for a basic connection, you must know enough about the protocol to issue valid commands. This knowledge is fairly advanced, and you won't be expected to do such things on the Linux+ exam!

You should be aware that although `telnet` is good for testing TCP connections, it's not useful for testing UDP or other types of connections. A few protocols, such as some versions of the Network File System (NFS), use UDP, so `telnet` isn't a useful diagnostic for them.



If you need to know a port number to use `telnet` as described here, the `/etc/services` file may be of use; it provides a mapping of port numbers to service names. Chapter 9 also provides this information for several common servers.

## Summary

Networking is very important to many modern Linux systems, which frequently function as servers or workstations on local networks. Networks operate by breaking data into individual packets in a manner that's dictated by the particular protocol stack in use by the system. Linux includes support for several protocol stacks, the most important of which is TCP/IP, the protocol stack on which the Internet is built. You can configure Linux for TCP/IP networking by using DHCP to automatically obtain an address, by entering the information manually, or by establishing a PPP link. You can do any of these things using text-mode or GUI tools, although the GUI tools aren't standardized across different distributions.

You may need to diagnose network problems. Tools such as `ifconfig`, `netstat`, `arp`, `ping`, and `traceroute` are useful in this task. Each of these tools provides its own type of data, which can help you isolate the source of a problem.

## Exam Essentials

**Determine appropriate network hardware for a Linux computer.** If the computer is to be used on an existing network, you must obtain a network card of a type that's compatible with that network, such as Ethernet or Token Ring. If you're building a new local network, Ethernet is the most common choice, although more exotic alternatives are also available and may be suitable in some specific situations. Hardware supporting 802.11 (Wi-Fi) protocols is appropriate if wireless access is necessary.

**Summarize how most network hardware is activated in Linux.** The `ifconfig` command brings up a network card, assigning it an IP address and performing other basic configuration tasks. Typically, this command is called in a SysV startup script, which may perform still more tasks as well, such as adding entries to the routing table. Wireless hardware requires use of the `iwconfig` command prior to the `ifconfig` call.

**Describe the information needed to configure a computer on a static IP network.** Four pieces of information are important: the IP address, the netmask (aka the network mask or subnet mask), the network's gateway address, and the address of at least one DNS server. The first two are required, but if you omit either or both of the latter two, you won't be able to connect to the Internet or use most DNS hostnames.

**Determine when using `/etc/hosts` rather than DNS makes the most sense.** The `/etc/hosts` file provides a static mapping of hostnames to IP addresses on a single computer. As such, maintaining this file on a handful of computers for a small local network is fairly straightforward, but when the number of computers rises beyond a few or when IP addresses change frequently, running a DNS server to handle local name resolution makes more sense.

**Explain what the `route` command accomplishes.** The `route` command displays or modifies the routing table, which tells Linux how to direct packets based on their destination IP addresses.

**Summarize how `ping` and `tracert` differ.** The `ping` command sends a simple packet to a target, waits for a reply, and reports on the total round-trip time. The `tracert` command is similar, but it traces the route of a packet step-by-step, enabling you to track the source of a network connectivity problem.

## Review Questions

1. Which types of network hardware does Linux support? (Choose all that apply.)
  - A. Token Ring
  - B. Ethernet
  - C. DHCP
  - D. Fibre Channel
2. Which of the following is a valid IPv4 address on a TCP/IP network?
  - A. 202.9.257.33
  - B. 63.63.63.63
  - C. 107.29.5.3.2
  - D. 98.7.104.0/24
3. Which of the following is *not* a Linux DHCP client?
  - A. pump
  - B. dhcpcd
  - C. dhcpd
  - D. dhclient
4. You try to set up a computer on a local network via a static TCP/IP configuration, but you lack a gateway address. Which of the following is true?
  - A. Because the gateway address is necessary, no TCP/IP networking functions will work.
  - B. TCP/IP networking will function, but you'll be unable to convert hostnames to IP addresses, or vice versa.
  - C. You'll be able to communicate with machines on your local network segment but not with other systems.
  - D. The computer won't be able to tell which other computers are local and which are remote.
5. Which of the following types of information is returned by typing **ifconfig eth0**? (Choose all that apply.)
  - A. The names of programs that are using **eth0**
  - B. The IP address assigned to **eth0**
  - C. The hardware address of **eth0**
  - D. The hostname associated with **eth0**

6. In what way do GUI network configuration tools simplify the network configuration process?
  - A. They're the only way to configure a computer using DHCP, which is an easier way to set networking options than static IP addresses.
  - B. They provide the means to configure PPPoE, which is easier to configure than DHCP or static IP addresses.
  - C. Once running, they provide easy-to-find labels for options, obviating the need to locate appropriate configuration files.
  - D. They're consistent across distributions, making it easier to find appropriate options on an unfamiliar distribution.
7. Which of the following is an advantage of IPv6 over IPv4?
  - A. IPv6 provides a much larger address space than IPv4.
  - B. IPv6 provides better support for legacy OSs than IPv4.
  - C. IPv6 supports remote text-mode logins with encryption.
  - D. IPv6 works with Wi-Fi connections; IPv4 doesn't.
8. Under what circumstances might you use the `iwconfig` utility?
  - A. You must diagnose problems on a Token Ring network.
  - B. You need to bring up or shut down an Ethernet network link.
  - C. You need to connect a Linux system to a new wireless network.
  - D. You must diagnose the failure of a DHCP client on a HIPPI network.
9. Which of the following utilities can bring up a network connection? (Choose all that apply.)
  - A. `ifconfig`
  - B. `netstat`
  - C. `ifup`
  - D. `ping`
10. Which file would you modify to give `/etc/hosts` priority over DNS lookups?
  - A. `/etc/hosts`
  - B. `/etc/dns.conf`
  - C. `/etc/resolv.conf`
  - D. `/etc/nsswitch.conf`
11. You want to permanently change the IP address of an Ubuntu computer with a fixed IP address on an Ethernet network. What file would you edit to do this job?
  - A. `/etc/network/eth0`
  - B. `/etc/network/interfaces`
  - C. `/etc/resolv.conf`
  - D. `/etc/hostname`

12. You want to permanently change the IP address of a Fedora computer with a fixed IP address on an Ethernet network. What file would you edit to do this job?
  - A. `/etc/sysconfig/network-scripts/ifcfg-eth0`
  - B. `/etc/network/ifcfg`
  - C. `/etc/system-config/ipaddr`
  - D. `/etc/hosts`
13. You type **arp -n** at a command prompt. What type of output will you see?
  - A. A summary of network packet errors
  - B. Routing table information
  - C. The mapping of IP addresses to MAC addresses
  - D. The IP address(es) of your name server(s)
14. Your computer has an IP address of 192.168.21.102, with a network mask of 255.255.255.0. You're able to ping 192.168.21.7 and 192.168.21.98, but not 192.168.27.3 or 10.78.21.102. If you know that all of these addresses are valid and the computers are turned on and connected to the network, what is the most probable cause of this problem?
  - A. The name server configuration is set incorrectly on 192.168.21.102.
  - B. The default route is set incorrectly on 192.168.21.102.
  - C. The DHCP servers must be activated on 192.168.27.3 and 10.78.21.102.
  - D. The netmask is set incorrectly on 192.168.21.102.
15. Which of the following programs can be used to perform a DNS lookup in interactive mode?
  - A. `nslookup`
  - B. `host`
  - C. `pump`
  - D. `ifconfig`
16. Which of the following entries are found in the `/etc/hosts` file?
  - A. A list of hosts allowed to remotely access this one
  - B. Mappings of IP addresses to hostnames
  - C. A list of users allowed to remotely access this host
  - D. Passwords for remote Web administration
17. You use **traceroute** to trace the path from your computer to 172.24.24.24. You discover, however, that starting with 10.109.73.7, your packets are being lost. What can you conclude?
  - A. The 10.109.73.7 router is down.
  - B. You need to rerun the test with the `-n` option to **traceroute**.
  - C. The 172.24.24.24 server is down.
  - D. No solid conclusions can be drawn.

18. Your computer is in the `example.com` domain, but you want to be able to contact the `neil.tranquility.luna.edu` and `buzz.tranquility.luna.edu` servers by typing `neil` or `buzz` as the hostnames, respectively. How can you accomplish this goal? (Choose all that apply.)
- A. Add the lines `host neil neil.tranquility.luna.edu` and `host buzz buzz.tranquility.luna.edu` to your Bash startup script.
  - B. Add entries for `neil` and `buzz`, linking them to their IP addresses, to `/etc/hosts`.
  - C. Add the line `search tranquility.luna.edu` to your `/etc/resolv.conf` file.
  - D. Add the line `nameserver tranquility.luna.edu` to your `/etc/resolv.conf` file.
19. Which of the following commands should you use to add to host `192.168.0.10` a default gateway to `192.168.0.1`?
- A. `route add default gw 192.168.0.10 192.168.0.1`
  - B. `route add default gw 192.168.0.1`
  - C. `route add 192.168.0.10 default 192.168.0.1`
  - D. `route 192.168.0.10 gw 192.168.0.1`
20. The `telnet` program can be used to help diagnose connection problems to many servers. For what class of servers is this untrue?
- A. Servers that use HTTP
  - B. Servers that use SMTP
  - C. Servers that use TCP
  - D. Servers that use UDP

## Answers to Review Questions

1. A, B, D. Ethernet is currently the most common type of network hardware for local networks. Linux supports it very well, and Linux also includes support for Token Ring and Fibre Channel network hardware. DHCP is a protocol used to obtain a TCP/IP configuration over a TCP/IP network. It's not a type of network hardware, but it can be used over hardware that supports TCP/IP.
2. B. IP addresses consist of four 1-byte numbers (0–255). They're normally expressed in base 10 and separated by periods. 63.63.63.63 meets these criteria. 202.9.257.33 includes one value (257) that's not a 1-byte number. 107.29.5.3.2 includes five 1-byte numbers. 98.7.104.0/24 is a network address—the trailing /24 indicates that the final byte is a machine identifier, and the first three bytes specify the network.
3. C. Option C, `dhcpcd`, is the Linux DHCP *server*. The others are all DHCP clients. Most distributions ship with just one or two of the DHCP clients.
4. C. The gateway computer is a router that transfers data between two or more network segments. As such, if a computer isn't configured to use a gateway, it won't be able to communicate beyond its local network segment. (If your DNS server is on a different network segment, name resolution via DNS won't work, although other types of name resolution, such as `/etc/hosts` file entries, will still work.)
5. B, C. When used to display information on an interface, `ifconfig` shows the hardware and IP addresses of the interface, the protocols (such as TCP/IP) bound to the interface, and statistics on transmitted and received packets. This command does *not* return information on programs using the interface or the hostname associated with the interface.
6. C. Once you know what tool to run in a distribution, it's usually not difficult to find the label for any given network configuration option in a GUI tool. You can configure DHCP and PPPoE in text mode (and the latter is arguably more complex than DHCP). GUI configuration tools, although they provide similar functionality, are not entirely consistent from one distribution to another.
7. A. The main motivating force behind IPv6 is to increase the address space of TCP/IP from the  $2^{32}$  (approximately 4 billion) addresses provided by IPv4, which is becoming an obstacle to further Internet expansion. Option B is backwards; older OSs are more likely to have good IPv4 support than good IPv6 support. Encryption is largely a feature of client and server software at the Application level, and software such as SSH provides encrypted text-mode login support even with IPv4, so option C is incorrect. (That said, IPv6 does have features that are intended to improve security.) Wi-Fi connections can be made with either IPv4 or IPv6, contrary to option D.
8. C. The `iwconfig` utility configures a Linux wireless (Wi-Fi) connection, so option C is correct. Options A, B, and D all refer to wired network hardware, for which `iwconfig` is useless.



9. A, C. The `ifconfig` command is Linux's basic tool for manually bringing up a network connection using options provided on the command line, while `ifup` is a utility that brings up a network connection based on the contents of the network configuration files. The `netstat` and `ping` commands are both useful network diagnostic tools, but they don't bring up network connections.
10. D. The `/etc/nsswitch.conf` file includes a line called `hosts` that specifies, in order, what tools to use for hostname resolution. If `files` appears before `dns` on this line, the `/etc/hosts` file will be used before DNS accesses are attempted. The `/etc/hosts` file itself doesn't control its own priority, contrary to option A. There is no standard `/etc/dns.conf` file. The `/etc/resolv.conf` file points Linux to its name servers, but it doesn't affect the priority of `/etc/hosts` vs. DNS lookups.
11. B. The `/etc/network/interfaces` file holds critical network configuration information for Ubuntu systems, including the computer's IP address if that's configured manually. The `/etc/network/eth0` file is fictitious. The `/etc/resolv.conf` file holds name server information. The `/etc/hostname` file holds the computer's hostname, not its IP address.
12. A. Option A specifies the correct file (although the 0 number may be different if the computer has several Ethernet interfaces). Options B and C are fictitious. Option C specifies the file that holds static mappings of hostnames to IP addresses for local use.
13. C. The `arp` utility returns data from the Address Resolution Protocol (ARP) table, which holds mappings of IP addresses to hardware (MAC) addresses. The `-n` option displays IP addresses as such rather than converted into hostnames. Network packet error information, as stated in option A, can be obtained, along with other information, from `ifconfig` but not from `arp`. Routing table information, as stated in option B, is usually obtained from `route` or `netstat -r`. Your name server(s) IP address(es), as in option D, are most easily obtained by displaying the contents of `/etc/resolv.conf`.
14. B. The two reachable systems are on the same network block (192.168.21.0/24), so their network traffic would not pass through a router. The two unreachable systems are on different network blocks, which means their traffic must pass through a router. This pattern suggests that there's a routing problem, as in option B. (Another possibility is that the router itself is down or misconfigured.) Since all the computers in the question were specified by IP address, name server configuration, as in option A, isn't an issue. Option C implies that DHCP servers are required to respond to ping requests, but this isn't so; DHCP servers deliver IP addresses to DHCP clients. Although a change in the netmask might plausibly enable the first system to contact 192.168.27.3, if the two systems are connected to the same physical wires, the 10.78.21.102 system is much too different in IP address to make a misconfigured netmask a plausible explanation for this problem, as option D suggests.
15. A. The `nslookup` program, though being phased out, offers an interactive mode that can be used for DNS lookups. The `host` program, though a replacement for `nslookup`, does not offer an interactive mode. `pump` is a DHCP client, while `ifconfig` is used for configuration of networking parameters and cards.

16. B. The `/etc/hosts` file holds mappings of IP addresses to hostnames, on a one-line-per-mapping basis. It does not list the users or other hosts allowed to remotely access this one, nor does it affect remote administration through a Web browser.
17. D. Although `traceroute` can be a useful tool for locating problems, it's imperfect. Routers can sometimes be configured to drop `traceroute` packets but not regular packets, so you can't really be sure if 10.109.73.7 is dropping packets by design or if it's down; both are possibilities, so no conclusion can be drawn.
18. B, C. The `/etc/hosts` file contains static mappings of hostnames to IP addresses, so adding entries as specified in option B will work (although these entries will need to be changed if these servers' IP addresses ever change). Option C's solution will also work, and it will continue to work if the servers' IP addresses change, provided that their DNS server is appropriately updated. Option A won't work because the `host` command is a DNS lookup tool; it doesn't create hostname aliases, as option A implies. Option D confuses the search and `nameserver` functions of `/etc/resolv.conf`. The `nameserver` line in this file specifies a DNS name server by IP address, so option D won't work.
19. B. To add a default gateway of 192.168.0.1, the command would be **`route add default gw 192.168.0.1`**. Specifying the IP address of the host system is not necessary and in fact will confuse the `route` command.
20. D. Telnet is a TCP protocol, which means that it cannot connect to servers that use UDP exclusively. Thus, option D is correct. HTTP and SMTP are both TCP-based protocols for which `telnet` can be a useful diagnostic tool. Broadly speaking, Telnet can help diagnose TCP protocols, although this is easier for some protocols than for others.

# Chapter 9

## Configuring Advanced Networking

---

### THE FOLLOWING COMPTIA OBJECTIVES ARE COVERED IN THIS CHAPTER:

- ✓ 3.9 Deploy and manage CUPS print services (enable and disable queues, Web management interface [port 631], Printing commands: `lpr`, `lp`, `lpq`, `lpstat`, `cancel`).
- ✓ 4.1 Identify common networking ports and the associated service (20, 21, 22, 23, 25, 53, 80, 110, 123, 143, 443, 631, 3306, `/etc/services`).
- ✓ 4.3 Implement configurations and/or configuration changes for the following (Packet filtering: `iptables`).
- ✓ 5.7 Deploy remote access facilities using the following (SSH: secure tunnels, SFTP, X11 forwarding, Key gen; VNC).



Chapter 8, “Configuring Basic Networking,” introduced the topic of network configuration, but it covered only the basics of network configuration and diagnostics. This chapter continues this introduction to networking by tackling several important topics. These include additional information on router configuration, network ports, and packet filtering (a security tool that enables you to drop, redirect, or modify packets based on a wide variety of criteria). These topics are particularly important if you want to configure a Linux system as a router—a system that directs data between two networks.

This chapter also covers a couple of important classes of network servers and clients: remote login tools and the Common Unix Printing System (CUPS) utility for printing. Both ordinary users and system administrators can use remote network access. CUPS is used for printing even on Linux systems with no regular network connections, but CUPS really shines on a network, since it enables seamless sharing of printers. Chapter 10, “Configuring Network Servers I,” and Chapter 11, “Configuring Network Servers II,” continue the examination of Linux networking by delving into a number of additional network server packages.



Objective 4.3 is covered partly in this chapter and partly in Chapter 8.

## Routing Between Networks

Chapter 8 provided information on router configuration, including the use of the `route` command to tell your computer about the local networks to which it’s attached and how to direct network traffic to computers on other networks.

A typical configuration involves just one local network and one router or gateway system; however, Linux can support much more advanced configurations. For instance, suppose a computer has two network cards, each of which is connected to a different network. The routing table on such a computer might resemble the following, as reported by `route`:

```
$ route -n
```

```
Kernel IP routing table
```

| Destination | Gateway       | Genmask       | Flags | Metric | Ref | Use | Iface |
|-------------|---------------|---------------|-------|--------|-----|-----|-------|
| 172.24.21.0 | 0.0.0.0       | 255.255.255.0 | U     | 0      | 0   | 0   | eth0  |
| 192.168.1.0 | 0.0.0.0       | 255.255.255.0 | U     | 0      | 0   | 0   | eth1  |
| 0.0.0.0     | 192.168.1.254 | 0.0.0.0       | UG    | 100    | 0   | 0   | eth1  |

This computer (let's call it *speaker*) is connected to two networks: 172.24.21.0/24, on `eth0`; and 192.168.1.0/24, on `eth1`. The latter network includes a default gateway system (192.168.1.254), which handles traffic addressed to all computers on networks other than those explicitly included in the routing table.

Ordinarily, a computer configured in this way can communicate with the computers on both its networks and, via the gateway computer, to additional systems (perhaps including the entire Internet). One additional step will turn the computer into a router, which can then pass traffic back and forth between the two networks. If *speaker* is configured as a router, the systems on the 172.24.21.0/24 network might refer to it as a router, enabling them to communicate with the 192.168.1.0/24 systems, as well as all the computers that 192.168.1.254 can contact. Likewise, systems on the 192.168.1.0/24 network can communicate with 172.24.21.0/24. To enable this configuration, you must modify a key file in the `/proc` filesystem:

```
echo "1" > /proc/sys/net/ipv4/ip_forward
```

This command enables IP forwarding. Permanently setting this option requires modifying a configuration file. Some distributions set it in `/etc/sysctl.conf`:

```
net.ipv4.ip_forward = 1
```



The `sysctl` utility and `sysctl.conf` are described in greater detail in Chapter 3, “Managing Processes and Editing Files.”

Other distributions use other configuration files and options, such as `/etc/sysconfig/sysctl` and its `IP_FORWARD` line. If you can't find it, try using `grep` to search for `ip_forward` or `IP_FORWARD`, or enter the command to perform the change manually in a local startup script.



You shouldn't configure a Linux system as a router unless you understand the consequences. Linking two networks is often desirable or even necessary, but it may have security implications, particularly if one network is private (such as a network in a small office) and the other is public (such as the Internet). You can use `iptables`, as described in the next section, to increase the security on a router. You may also want to apply security software to all the computers on the smaller network and take extra care to secure the router itself. Chapter 12, “Securing Linux,” provides a starting point for this task.

## Firewall Configuration

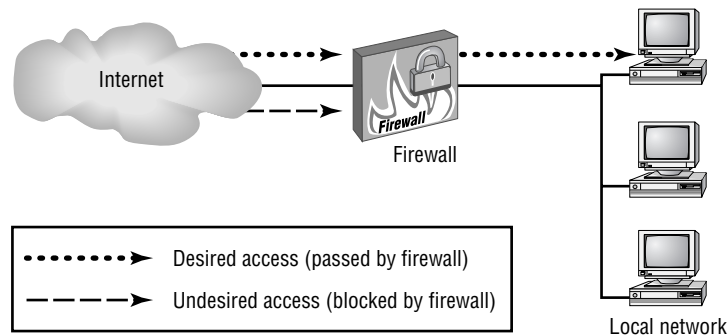
The first line of defense in network security is a *firewall*. This is a computer that restricts access to other computers, or it's software that runs on a single computer to protect it alone. Broadly speaking, two types of firewalls exist: *packet-filter firewalls*, which work by blocking

or permitting access based on low-level information in individual data packets, such as source and destination IP addresses and ports, and *proxy filters*, which partially process a transaction, such as a Web page access, and block or deny access based on high-level features in this transaction, such as the filename of an image in the Web page. This chapter describes Linux's packet-filter firewall tools, which can be very effective at protecting a single computer or an entire network against certain types of attack.

## Where a Firewall Fits in a Network

Traditionally, firewalls have been routers that block undesired network transfers between two networks. Typically, one network is a small network under one management, and the other network is much larger, such as the Internet. Figure 9.1 illustrates this arrangement. (More complex firewalls that use multiple computers are also possible.) Dedicated external firewalls are available, and they can be good investments in many cases. In fact, it's possible to turn an ordinary computer into such a device by using Linux—either with a special-purpose distribution like the Linux Embedded Appliance Firewall (<http://leaf.sourceforge.net>) or by using an ordinary distribution and configuring it as a router with firewall features.

**FIGURE 9.1** Firewalls can selectively pass some packets but not others, using assorted criteria.



As described in more detail shortly, servers operate by associating themselves with particular network ports. Likewise, client programs bind to ports, but client port bindings aren't standardized. Packet filter firewalls block access by examining individual network packets and determining whether to let them pass based on the source and destination port number, the source and destination IP address, and possibly other low-level criteria, such as the network interface in a computer with more than one. For instance, in Figure 9.1, you might run a Samba file server internally, but outside computers have no business accessing that server. Therefore, you'd configure the firewall to block external packets directed at the ports used by Samba.

In addition to running a firewall on a router that serves an entire network, it's possible to run a firewall on an individual system. This approach can provide added protection to

a sensitive computer, even if an external firewall protects that computer. It's also useful on computers that don't have the protection of a separate firewall, such as many broadband-connected systems.

## Linux Firewall Software

Linux uses the `ipfwadm`, `ipchains`, and `iptables` tools to configure firewall functions. These tools are designed for the 2.0.x, 2.2.x, and 2.4.x kernels, respectively. The 2.6.x kernels continue to use the `iptables` tool. (The 2.4.x and later kernel series include the ability to use the older tools, but only as a compile-time option.) You can configure a firewall in any of several ways:

**Manually** You can read up on the syntax of the tool used to configure your kernel and write your own script. This approach is described in the upcoming section “Using *iptables*.”



For more information on this approach, consult a book on the subject, such as Michael Rash's *Linux Firewalls: Attack Detection and Response with iptables, psad, and fwsnort* (No Starch Press, 2007) or Steve Suehring and Robert Ziegler's *Linux Firewalls, 3rd Edition* (Novell Press, 2005).

**With the help of a GUI configuration tool** A few GUI configuration tools are available for Linux firewall configuration, such as Firestarter (<http://firestarter.sourceforge.net>) and Guarddog (<http://www.simonzone.com/software/guarddog>). Linux distributions often incorporate such tools as well, although the distribution-provided tools are often very simple. These tools let you specify certain basic information, such as the network port and the client and server protocols you want to allow, and they generate firewall scripts that can run automatically when the system boots.

If you use a GUI tool, be sure it supports the firewall tool your kernel requires—some older tools were written for `ipchains` only or were designed for non-Linux OSs, but you're almost certainly using a 2.6.x kernel and so will want something that supports `iptables`. Also, you shouldn't consider a firewall to be perfect protection. You might create a configuration that contains flaws, or flaws might exist in the Linux kernel code that actually implements the firewall rules.



One of the advantages of a firewall, even to protect just one computer, is that it can block access attempts to *any* server. Most other measures are more limited. For instance, TCP Wrappers (described in Chapter 4, “Managing System Services”) protects only servers configured to be run via TCP Wrappers from `inetd`, and passwords are good only to protect the servers that are coded to require them.



## Common Server Ports

Most packet filter firewalls use the server program's port number as a key feature. For instance, a firewall might block outside access to the SMB/CIFS ports used by Samba but let through traffic to the SMTP mail server port. In order to configure a firewall in this way, of course, you must know the port numbers. Linux systems contain a file, `/etc/services`, that lists service names and the ports with which they're associated. Lines in this file look something like this:

```
ssh 22/tcp # SSH Remote Login Protocol
ssh 22/udp # SSH Remote Login Protocol
telnet 23/tcp
24 - private
smtp 25/tcp
```

The first column contains a service name (`ssh`, `telnet`, or `smtp` in this example). The second column contains the port number and protocol (such as `22/tcp`, meaning TCP port 22). Anything following a hash mark (`#`) is a comment and is ignored. The `/etc/services` file lists port numbers for both TCP and UDP ports. Typically, a single service is assigned the use of the same TCP and UDP port numbers (as in the `ssh` service in this example), although most protocols use just one or the other. When configuring a firewall, it's generally best to block both TCP and UDP ports; this ensures you won't accidentally block the wrong port type.

Table 9.1 summarizes the port numbers used by the most important protocols run on Linux systems. This list is, however, incomplete; it hits only some of the most common protocols. In fact, even `/etc/services` is incomplete and may need to be expanded for certain obscure servers. (The documentation for such servers describes how to do so, if necessary.)

**TABLE 9.1** Port Numbers Used by Some Common Protocols

| Port Number | TCP/UDP     | Protocol | Example Server Programs        |
|-------------|-------------|----------|--------------------------------|
| 20 and 21   | TCP         | FTP      | ProFTPD, WU FTPd               |
| 22          | TCP         | SSH      | OpenSSH, lsh                   |
| 23          | TCP         | Telnet   | in.telnetd                     |
| 25          | TCP         | SMTP     | sendmail, Postfix, Exim, qmail |
| 53          | TCP and UDP | DNS      | BIND, dnsmasq                  |
| 67          | UDP         | DHCP     | DHCP, dnsmasq                  |
| 69          | UDP         | TFTP     | in.tftpd                       |
| 80          | TCP         | HTTP     | Apache, thttpd                 |



**TABLE 9.1** Port Numbers Used by Some Common Protocols *(continued)*

| Port Number | TCP/UDP     | Protocol               | Example Server Programs            |
|-------------|-------------|------------------------|------------------------------------|
| 88          | TCP         | Kerberos               | MIT Kerberos, Heimdal              |
| 109 and 110 | TCP         | POP (versions 2 and 3) | UW IMAP, Cyrus IMAP                |
| 111         | TCP and UDP | Portmapper             | NFS, NIS, other RPC-based services |
| 113         | TCP         | auth/ident             | identd                             |
| 119         | TCP         | NNTP                   | INN, Leafnode                      |
| 123         | UDP         | NTP                    | NTP                                |
| 137         | UDP         | NetBIOS Name Service   | Samba                              |
| 138         | UDP         | NetBIOS Datagram       | Samba                              |
| 139         | TCP         | NetBIOS Session        | Samba                              |
| 143         | TCP         | IMAP 2                 | UW IMAP, Cyrus IMAP                |
| 177         | UDP         | XDMCP                  | XDM, KDM, GDM                      |
| 220         | TCP         | IMAP 3                 | UW IMAP, Cyrus IMAP                |
| 389         | TCP         | LDAP                   | OpenLDAP                           |
| 443         | TCP         | HTTPS                  | Apache                             |
| 445         | TCP         | Microsoft DS           | Samba                              |
| 514         | UDP         | Syslog                 | syslogd                            |
| 515         | TCP         | Spooler                | BSD LPD, LPRng, cups-lpd           |
| 631         | TCP         | IPP                    | CUPS                               |
| 636         | TCP         | LDAPS                  | OpenLDAP                           |
| 749         | TCP         | Kerberos Admin         | MIT Kerberos, Heimdal              |
| 3306        | TCP         | SQL                    | MySQL                              |
| 5800–5899   | TCP         | VNC via HTTP           | RealVNC, TightVNC                  |

**TABLE 9.1** Port Numbers Used by Some Common Protocols (*continued*)

| Port Number | TCP/UDP | Protocol | Example Server Programs |
|-------------|---------|----------|-------------------------|
| 5900–5999   | TCP     | VNC      | RealVNC, TightVNC       |
| 6000–6099   | TCP     | X        | X.org-X11, XFree86      |



Table 9.1 shows the ports used by the *servers* for the specified protocols. In most cases, clients can and do use other port numbers to initiate connections. For instance, a mail client might use port 43411 on `client.pangaea.edu` to connect to port 143 on `mail.pangaea.edu`. Client port numbers are assigned by the kernel on an as-needed basis, so they aren't fixed. (Clients can request specific port numbers, but this practice is rare.)

One key distinction in TCP/IP ports is that between *privileged ports* and *unprivileged ports*. The former have numbers below 1024. Unix and Linux systems restrict access to privileged ports to root. The idea is that a client can connect to a privileged port and be confident that the server running on that port was configured by the system administrator and can therefore be trusted. Unfortunately, on today's Internet, this trust would be unjustified based solely on the port number, so this distinction isn't very useful. Port numbers above 1024 may be opened by ordinary user programs.



ISPs typically block access to many common server ports on residential accounts. This practice prevents home users, and even some small businesses, from running their own Web, e-mail, and other common servers. Users can still access outside servers' ports in most cases, although as a spam-reduction measure, ISPs sometimes block outgoing port 25 access except to their own mail servers.

## Using *iptables*

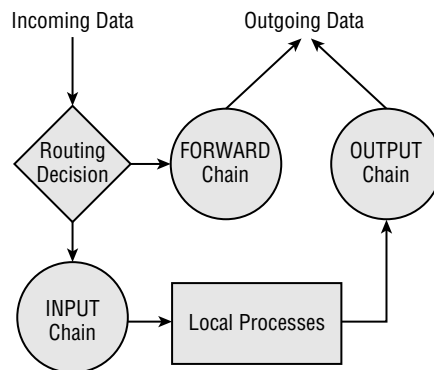
The *iptables* program is the utility that manages firewalls on recent Linux kernels (from 2.4.x through at least 2.6.x). Although these kernels can also use the older *ipchains* tool when so configured using kernel compile-time options, *iptables* is the more flexible tool and is therefore the preferred way of creating and managing packet-filter firewalls.

When using *iptables*, you should first understand how Linux's packet filter architecture works—you can create several types of rules, which have differing effects, so understanding how they interact is necessary before you begin creating rules. Actually creating the rules requires understanding the *iptables* command syntax and options. Finally, it's helpful to look at a sample firewall script and to know how it's installed and called by the system.

## The Linux Packet Filter Architecture

In the 2.4.x and later kernels, Linux uses a series of “tables” to process all network packets it generates or receives. Each table consists of several “chains,” which are series of pattern-matching rules—when a packet matches a rule, the rule can discard the packet, forward it to another chain, or accept the packet for local delivery. Figure 9.2 illustrates the `filter` table, which is the one you normally modify when creating a firewall. Other tables include the `nat` table, which implements network address translation (NAT) rules, and the `mangle` table, which modifies packets in specialized ways.

**FIGURE 9.2** Linux uses a series of rules, which are defined in chains that are called at various points during processing, to determine the fate of network packets.



As shown in Figure 9.2, the `filter` table consists of three chains: `INPUT`, `OUTPUT`, and `FORWARD`. These chains process traffic directed to local programs, generated by local programs, and forwarded through a computer that’s configured as a router, respectively. You can create rules independently for each chain. For instance, consider a rule that blocks all access directed at port 80 (the HTTP port, used by Web servers) on any IP address. Applied to the `INPUT` chain, this rule blocks all access to a Web server running on the local computer but doesn’t affect outgoing traffic or traffic that’s forwarded if the computer is configured as a router. Applied to the `OUTPUT` chain, this rule blocks all outgoing traffic directed at Web servers, effectively rendering Web browsers useless, but it doesn’t affect incoming traffic directed at a local Web server or traffic forwarded by a router. Applied to the `FORWARD` chain, this rule blocks HTTP requests that might otherwise be forwarded by a computer that functions as a router, but it doesn’t affect traffic from local Web browsers or to local Web servers.

Much of the task of creating a firewall involves deciding which chains to modify. Generally speaking, when you want to create a separate firewall computer (as illustrated in Figure 9.1), you modify the `FORWARD` chain (to protect the computers behind the firewall) and the `INPUT` chain (to protect the firewall system itself). When implementing a firewall to protect a server or workstation, you modify the `INPUT` chain and perhaps the `OUTPUT` chain. Blocking output packets can have the effect of preventing abuse of other systems or use of protocols you don’t

want being used. For instance, you might block outgoing traffic directed to a remote system's port 23, effectively disallowing use of Telnet clients on the system you're configuring.

All of the chains implement a default policy. This policy determines what happens to a packet if no rule explicitly matches it. The default for a default policy is **ACCEPT**, which causes packets to be accepted. This policy is sensible in low-security situations, but for a more secure configuration, you should change the default policy to **DROP** or **REJECT**. The former causes packets to be ignored. To the sender, it looks as if a network link is down. The **REJECT** policy causes the system to actively refuse the packet, which looks to the sender as if no server is running on the targeted port. This option requires explicit kernel support. Both **DROP** and **REJECT** have their advantages. **DROP** reduces network bandwidth use and reduces the system's visibility on the network, whereas **REJECT** can improve performance for some protocols, such as **auth/ident**, which may retry a connection in the event a packet is lost. Using either **DROP** or **REJECT** as a default policy means that you must explicitly open ports you want to use. This is more secure than using a default policy of **ACCEPT** and explicitly closing ports, because you're less likely to accidentally leave a port open when it should be closed. Setting a default policy is described in the next section.

## Creating Firewall Rules

To create firewall rules, you use the **iptables** command. You should probably start with the **-L** option, which lists the current configuration:

```
iptables -L -t filter
Chain INPUT (policy ACCEPT)
target prot opt source destination

Chain FORWARD (policy ACCEPT)
target prot opt source destination

Chain OUTPUT (policy ACCEPT)
target prot opt source destination
```

The **-t filter** part of this command specifies that you want to view the **filter** table. This is actually the default table, so you can omit this part of the command, if you like. The result is a list of the rules that are defined for the specified (or default) table. In this case, no rules are defined, and the default policy is set to **ACCEPT** for all three chains in the table. This is a typical starting point, although depending on your distribution and your installation options, it's possible yours will have rules already defined. If so, you should track down the script that sets these rules and change or disable it. Alternatively, or if you just want to experiment, you can begin by flushing the table of all rules by passing **-F CHAIN** to **iptables**, where **CHAIN** is the name of the chain. You can also use **-P CHAIN POLICY** to set the default policy:

```
iptables -t filter -F FORWARD
iptables -t filter -P FORWARD DROP
```

These two commands flush all rules from the FORWARD chain and change the default policy for that chain to DROP. Generally speaking, this is a good starting point when configuring a firewall, although using REJECT rather than DROP has its advantages, as described earlier. You can then add rules to the chain, each of which matches some selection criterion. To do so, you use an iptables command of this form:

```
iptables [-t table] -A CHAIN selection-criteria -j TARGET
```

When modifying the filter table, you can omit the `-t table` option. The *TARGET* is the policy target, which can take the same values as the default policy (typically ACCEPT, REJECT, or DROP). In most cases, you'll use ACCEPT when the default policy is REJECT or DROP and you'll use REJECT or DROP when the default policy is ACCEPT. *CHAIN* is, as you might expect, the chain to be modified (INPUT, OUTPUT, or FORWARD for the filter table). Finally, *selection-criteria* can be one or more of several options that enable you to match packets by various rules, as summarized in Table 9.2.

TABLE 9.2 iptables Selection Criteria Options and Effects

| Option                         | Option Abbreviation | Effect                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|--------------------------------|---------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| --protocol <i>name</i>         | -p <i>name</i>      | This option lets you specify the low-level protocol used. You pass the protocol name (tcp, udp, icmp, or all) to match packets of the specified protocol type. The all name matches all protocol types, though.                                                                                                                                                                                                                                                |
| --source-port <i>port</i>      | --sport <i>port</i> | This option matches packets that originate from the port number that you specify. (You can also provide a list of port numbers by separating them with colons, as in 1024:2048 to specify ports from 1024 to 2048, inclusive.) Note that the originating port number is the port number for the server program for packets that come from the server system, but it's the port number used by the client program for packets that come from the client system. |
| --destination-port <i>port</i> | --dport <i>port</i> | This option works much like the --source-port option, but it applies to the destination of the packet.                                                                                                                                                                                                                                                                                                                                                         |
| --source <i>ipaddr</i>         | -s <i>ipaddr</i>    | This option filters on the source IP address. You can specify either a single IP address or an entire subnet by appending the netmask as a number of bits, as in -s 172.24.1.0/24.                                                                                                                                                                                                                                                                             |

**TABLE 9.2** iptables Selection Criteria Options and Effects *(continued)*

| Option                       | Option Abbreviation | Effect                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|------------------------------|---------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| --destination <i>ipaddr</i>  | -d <i>ipaddr</i>    | This option works just like the --source option, but it filters based on a packet's destination address.                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| --in-interface <i>iface</i>  | -I <i>iface</i>     | You can use the interface on which the packet arrives with this option, which accepts an interface name as an argument. For instance, -I eth0 matches packets that arrive on the eth0 interface. This option works with the INPUT and FORWARD chains, but not with the OUTPUT chain.                                                                                                                                                                                                                                                                                      |
| --out-interface <i>iface</i> | -o <i>iface</i>     | This option works much like the --in-interface option, but it applies to the interface on which packets will leave the computer. As such, it works with the FORWARD and OUTPUT chains, but not with the INPUT chain.                                                                                                                                                                                                                                                                                                                                                      |
| --match <i>module</i>        | -m <i>module</i>    | This option specifies a module used to expand options. One common module is state, which enables stateful packet inspection, as used by --state.                                                                                                                                                                                                                                                                                                                                                                                                                          |
| --state <i>state</i>         | None                | Network connections have states—they can be used to initiate a new connection, continue an existing connection, be related to an existing connection (such as an error message), or be potentially forged. This option can match based on these states, using codes of NEW, ESTABLISHED, RELATED, or INVALID. You must precede this option with the -m state option on the same iptables command line. This feature is most useful in blocking connection attempts to unprivileged ports, thus denying miscreants the ability to run unauthorized servers on those ports. |

You can combine multiple items to filter based on several criteria. For instance, in a default-deny configuration, you can open traffic to TCP port 445 from the 172.24.1.0/24 network with a single command:

```
iptables -A INPUT -p tcp --dport 445 -s 172.24.1.0/24 -j ACCEPT
```

In this case, the *selection-criteria* consist of three rules: -p tcp, --dport 445, and -s 172.24.1.0/24. Packets that match *all* of these rules will be accepted; those that fail to match even a single rule will be denied (assuming this is the default configuration), unless they match some other rule in the chain.

A complete chain is created by issuing multiple `iptables` commands, each of which defines a single rule. You can then view the result by typing `iptables -L`, as described earlier.

## A Sample *iptables* Configuration

Because `iptables` creates a complete firewall configuration only through the use of multiple calls to the utility, Linux packet-filter firewalls are frequently created via shell scripts that repeatedly call `iptables`. (Chapter 2, “Using Text-Mode Commands,” introduces shell scripts, so review it if you need more information on the basics of creating a script.) These scripts may be called as SysV startup scripts or in some other way as part of the startup procedure. For learning purposes, you may want to create a script that’s not called in this way, though. Listing 9.1 shows a sample script that demonstrates the key points of firewall creation.

### Listing 9.1: Sample Linux Firewall Script

```
#!/bin/bash

iptables -F INPUT
iptables -F FORWARD
iptables -F OUTPUT

iptables -P INPUT DROP
iptables -P FORWARD DROP
iptables -P OUTPUT DROP

Let traffic on the loopback interface pass
iptables -A OUTPUT -d 127.0.0.1 -o lo -j ACCEPT
iptables -A INPUT -s 127.0.0.1 -i lo -j ACCEPT

Let DNS traffic pass
iptables -A OUTPUT -p udp --dport 53 -j ACCEPT
iptables -A INPUT -p udp --sport 53 -j ACCEPT

Let clients' TCP traffic pass
iptables -A OUTPUT -p tcp --sport 1024:65535 -m state \
 --state NEW,ESTABLISHED,RELATED -j ACCEPT
iptables -A INPUT -p tcp --dport 1024:65535 -m state \
 --state ESTABLISHED,RELATED -j ACCEPT

Let local connections to local SSH server pass
iptables -A OUTPUT -p tcp --sport 22 -d 172.24.1.0/24 -m state \
 --state ESTABLISHED,RELATED -j ACCEPT
iptables -A INPUT -p tcp --dport 22 -s 172.24.1.0/24 -m state \
 --state NEW,ESTABLISHED,RELATED -j ACCEPT
```

Listing 9.1 consists of three broad parts. The first three calls to `iptables` clear out all preexisting firewall rules. This is particularly important in a script that you're creating or debugging because you don't want to simply add new rules to existing ones—the result would likely be a confusing mishmash of old and new rules. The next three calls to `iptables` set the default policy to `DROP` on all three chains. This is a good basic starting point for a firewall. The remaining calls to `iptables` configure Linux to accept specific types of traffic:

**Loopback traffic** The script sets the system to accept traffic to and from the loopback interface (that is, 127.0.0.1). Certain Linux tools expect to be able to use this interface, and because it's purely local, the security risk in accepting such traffic is very slim. Note that the lines that enable this access use both the IP address (via the `-d` and `-s` options) and the `lo` interface name (via the `-o` and `-i` options). This configuration protects against spoofing the loopback address—an attacker pretending to be 127.0.0.1 from another computer. This configuration, like most `iptables` configurations, requires two `iptables` rules: one to enable incoming traffic and one to enable outgoing traffic.

**DNS traffic** The second block of rules enables UDP traffic to and from port 53, which handles DNS. A configuration like this one is necessary on most systems to enable the computer to use its local DNS server. You could strengthen this configuration by specifying only your local DNS server's IP address. (If you have multiple DNS servers, you'd need one pair of rules for each one.)

**Client traffic** Listing 9.1 enables TCP packets to be sent from unprivileged ports (those used by client programs) to any system. This configuration uses stateful inspection to enable new, established, or related outgoing traffic but to allow only established or related incoming traffic. This configuration effectively blocks the ability to run servers on unprivileged ports. Thus, an intruder or malicious authorized user won't be able to log into an unauthorized server that runs on such a port—at least, not without `root` access to change the configuration.

**SSH server traffic** The final block of options enables access to the SSH server (TCP port 22). This access, though, is restricted to the 172.24.1.0/24 network (presumably the local network for the computer). This configuration uses stateful packet inspection to outgoing traffic from the SSH server for established and related data, but not for new or invalid packets. Incoming packets to the server are permitted for new, existing, or related traffic, but not for invalid packets.

A configuration such as the one in Listing 9.1 is suitable for a workstation that runs an SSH server for remote administration but that otherwise runs no servers. For a computer that runs many servers, you would need to add several additional blocks of rules similar to the SSH block, each one customized for a particular server. For a dedicated router with firewall features, the emphasis would be on the `FORWARD` chain rather than the `INPUT` and `OUTPUT` chains, although such a system would likely need to perform some `INPUT` and `OUTPUT` chain configuration to support its own administration and use.



# Managing Remote Logins

Linux's multiuser, multitasking nature invites remote access: you can use both text-mode and GUI tools to log in remotely and use the computer, both as an ordinary user and as a system administrator. Even if a computer is primarily a user workstation, remote access can be very convenient for system administration purposes—you as a system administrator can log into another user's computer to diagnose problems, install software, and so on.



Chapter 11 describes GUI remote access tools that are particularly useful for interacting with Windows systems. This chapter emphasizes text-mode and Linux/Unix GUI remote access tools.

## Setting Up a Remote Access Server

One of the oldest remote access protocols around is Telnet, and all major Linux distributions ship with a Telnet server, which is typically called `telnetd` or `in.telnetd`. This file may be distributed in a package called `telnet`, `telnet-server`, or something else. Telnet servers are very simple and therefore require no configuration beyond basic installation. They're normally launched from `inetd` or `xinetd`, which are programs that start other servers on an as-needed basis. Chapter 4 describes `inetd` and `xinetd` in more detail.

Unfortunately, Telnet suffers from serious security problems—it sends passwords (and all other data) unencrypted across the network. Therefore, the SSH protocol has emerged as a more secure replacement for Telnet. The most popular SSH package in Linux is OpenSSH (<http://www.openssh.com>). SSH typically comes in at least two packages: a client and a server. There may also be a “common” package and support libraries.

Once all the required packages are installed and the server is running, the default SSH configuration tends to work well. If necessary, though, you can fine-tune it. The normal SSH server configuration file is `/etc/ssh/sshd_config`. (There's also an `/etc/ssh/ssh_config` file that controls the SSH client.)



Some SSH packages come configured to allow root to log in directly. Even with the password encryption provided by SSH, this is inadvisable because it makes it too easy for somebody who has obtained the root password through other means to break into your system. To plug this security hole, change the `PermitRootLogin` option in `sshd_config` to `no`. Users who need to perform superuser tasks remotely can still log in as ordinary users and then use `su` to obtain the necessary privileges. This requires an outsider to have *two* passwords in order to do serious damage to the system.

SSH supports two major protocol versions: 1 and 2. These are enabled using the `Protocol` line in `/etc/ssh/sshd_config`. For best security, *only* protocol version 2 should be supported:

```
Protocol 2
```

If you enable protocol version 1 in addition to version 2 (via a `2,1` or `1,2` option on the `Protocol` line), the server will accept logins from clients using protocol version 1, which is less secure than version 2. A configuration that accepts only version 1 is also possible but is even worse than a configuration that accepts both protocols.

In addition to functioning as a text-mode remote-access tool, SSH permits *port forwarding*, in which another protocol's traffic travels over the SSH connection. This is most easily managed in conjunction with X, as described in “Using X Programs Remotely.” To enable X port forwarding, look for the following line in `/etc/ssh/sshd_config`:

```
X11Forwarding yes
```

If this option reads `no` rather than `yes`, change it to read `yes`. The forwarding of other TCP protocols is controlled via another option:

```
AllowTcpForwarding yes
```

The default value for this option is `yes`. If enabled, users may forward ports from a client to the server using SSH.

If you want to be able to use the `sftp` program (described in the next section) for file transfer, be sure that your `sshd_config` file includes a line like the following:

```
Subsystem sftp /usr/lib/openssh/sftp-server
```

The path to the `sftp-server` program might not be the same as the one shown in this example, but a similar line should exist. If it doesn't, add one, but be sure that the `sftp-server` program exists on your system and that you point to it using the correct path.

## Using Text-Mode Logins

You can use SSH to access one Linux computer from another system—even from a computer running another OS, such as Windows or Mac OS. Typically, you log in using a regular user account. You may then use `su` or `sudo` to acquire superuser privileges, if you want to perform administrative tasks. Thereafter, you can do almost anything you could do from a text-mode login at the console.



Telnet passes all data in an unencrypted form. This means that both your ordinary user's login password and the root password you enter in conjunction with `su` might be intercepted by an unscrupulous individual on the source, destination, or any intervening network. For this reason, it's best not to use Telnet for remote administration. For that matter, if it's possible, you should totally avoid using Telnet. SSH encrypts all the data that pass between two systems, so it is a much better choice for remote administration.

To use SSH for remote access, you type the SSH client's name followed by the server name. SSH passes your current username to the server, which attempts to use the same username to authenticate you. If you want to use a different username on the server than on your current system, you should include the `-l username` parameter on the command line or prepend the username to the hostname with an at-sign (`@`), as follows:

```
$ ssh ecernan@apollo.luna.edu
ecernan@apollo.luna.edu's password:
Last login: Tue Sep 30 10:43:37 from gemini.luna.edu
[ecernan@apollo ecernan]$
```



The first time you make a connection to a given server, you may see a message informing you that the authenticity of the server can't be verified. The message goes on to display a code associated with the server. If you want to continue connecting, type **yes** in response to the query.

You may omit the username and at-sign if your username is the same on both systems. Once you've logged in with SSH, you can use the system much as you would from the console—by typing text-mode commands, editing files with text-mode editors, and so on. Because SSH encrypts all data, it's extremely unlikely that your original password, or the password you type when you use `su`, will be usable to anybody who intercepts the data stream.

A variant on using SSH for remote login access is to use the protocol for running a specific program. Simply type the command you want to run at the end of the `ssh` command line, as in `ssh apollo.luna.edu cat /etc/fstab` to view the contents of `/etc/fstab` on `apollo.luna.edu`.

SSH does further duty as a secure file transfer protocol. On the client side, you can use the `scp` or `sftp` commands to handle this job. The `scp` command is similar to `cp`, but it operates across an SSH link. You add the hostname and, optionally, remote username to either the source or destination filename, as follows:

```
$ scp gemini.txt gene@apollo.luna.edu:
```

...or...

```
$ scp gene@apollo.luna.edu:xvii.txt ./
```

The first command copies the local file `gemini.txt` to the home directory of the user `gene` on `apollo.luna.edu`. The second command copies `xvii.txt` from `apollo.luna.edu` to the local computer. If your username on both systems is the same, you may omit the username and at-sign (`@`) from the command.



It's easy to forget the colon (:) at the end of the machine name when using `scp`. If you do, `scp` works much like `cp` when given the same arguments—it will copy the file locally, so you'll end up with a local file named after the remote username and computer hostname, such as `gene@apollo.luna.edu`. This is usually a minor inconvenience, but if you don't verify that a file has been transferred, it could turn into a major inconvenience at a later time.

The `sftp` program acts much like the more common `ftp` program: type the command name followed by the remote hostname (optionally preceded by a username). The computer then presents an `sftp>` prompt, at which you can type commands similar to those used by the `ftp` program. Example commands include `get` (to retrieve a file), `put` (to put a file), `dir` or `ls` (to view a directory listing), `cd` (to change into a directory on the remote system), `lcd` (to change into a directory on the local system), and `exit` or `quit` (to terminate the program). Chapter 11 describes the `ftp` client program in more detail. The `sftp` program requires that the server support its variant protocol, as described earlier.

## Generating SSH Keys

SSH works, in part, by using a series of *keys*, which are large numbers used to encrypt and decrypt data. Keys may also be used as a means of identification. Although SSH generates keys automatically so that you can use SSH without explicitly configuring keys, understanding SSH keys will enable you to perform tasks with the protocol you wouldn't otherwise be able to do.

The SSH server stores its keys in `/etc/ssh`, in a series of files called `ssh_host_rsa_key`, `ssh_host_dsa_key`, and these same filenames with `.pub` extensions. The files with `.pub` extensions hold *public keys*—keys that are given to other computers as a means of decrypting data that's been encrypted with the server's *private keys* (the keys stored in files without `.pub` extensions).



Private keys are extremely sensitive. They should be stored with 0600 (`-rw-----`) or more restrictive permissions, and they should never be sent to other parties. If a miscreant obtains your private key, that individual can impersonate you. Public keys are not very sensitive, so they can usually be read by anybody on the local computer, and they're frequently distributed to other computers as a normal part of SSH operation.

Individual users can also have SSH private and public keys. The `ssh` client program can automatically generate some keys as needed; however, you can alter the way SSH authenticates users by manually generating a new key. Using this method, either you'll use a passphrase instead of your normal password or you'll eliminate the need to type a password when you use SSH. To generate such a key, follow these steps:

1. Log into the SSH client system.

2. Generate an SSH version 2 key by typing the following command:

```
$ ssh-keygen -q -t rsa -f ~/.ssh/id_rsa -C '' -N ''
```

Omitting the `-N ''` part of this command causes `ssh-keygen` to prompt for a passphrase, which you will then have to use whenever you access a remote system. Using `-N ''` or pressing the Enter key in response to the passphrase prompt will eliminate the need to type a password or passphrase when connecting to remote systems.

3. Copy the `~/.ssh/id_rsa.pub` file generated in step 2 to your account on the SSH server system. Don't overwrite the file of this name on the server, though; give it a unique name.
4. Log into the server. If you use SSH, you'll have to enter a password.
5. Change to the `~/.ssh` directory on the server.
6. Add the client's `id_rsa.pub` file to the file that holds authorized keys on the server. This file may be called `authorized_keys` or `authorized_keys2`, depending on the SSH version. If the file doesn't exist, create it (or rename the copied `id_rsa.pub` file). You can combine the files using a text editor or by using a command like the following:

```
$ cat id_rsa.client >> authorized_keys
```

Once this task is done, you should be able to access the server using `ssh`, `scp`, and `sftp` from the client without typing a password (or by typing a passphrase instead of a password). The key you generated serves as a password substitute. This configuration can be extremely convenient, particularly if you want to use scripts or other automated tools to transfer files or otherwise use SSH features. If you access multiple servers, you should repeat steps 3–6 for each server, but you should *not* repeat step 2! Doing so will generate a new private key, invalidating the one you've copied to the first server.



The private key generated in this way is extremely sensitive. If somebody steals your private key, that person will be able to log into any server that's been given the matching public key. For this reason, you might want to think twice before using this method on computers that might be compromised, such as laptops that might be stolen.

## Using X Programs Remotely

Linux's GUI environment, the X Window System (or X for short), is unusual in that it's fully network-enabled. Using nothing but the normal X software and Linux network configuration, you can run an X program on one computer while sitting at another computer, using the second computer's monitor, keyboard, and mouse. In fact, one of these systems can run a Unix OS that's not Linux. You can even run an X server on a Windows, OS/2, or other completely non-Unix system, or on a system with a different class of CPU than the Linux system.



Although most people think of clients as running on the computers at which they sit and servers as running on remote systems, this isn't true of X. In X, the server runs on the system local to the user. To make sense of this, think of it from the program's point of view. To a word processor, the display and keyboard are services to be used, much like a network-accessible printer.

Suppose that your local network contains two machines. The computer called *zeus* is a powerful machine that hosts important programs, like a word processor and data analysis utilities. The computer called *apollo* is a much less powerful system, but it has an adequate monitor and keyboard. Therefore, you want to sit at *apollo* and run programs that are located on *zeus*. Both systems run Linux. To accomplish this task, follow these steps:

1. Log into *apollo* and, if it's not already running X, start it.
2. Open a terminal (such as an *xterm*) on *apollo*.
3. Type **xhost +zeus** in *apollo*'s terminal. This command tells *apollo* to accept for display in its X server data that originates on *zeus*.
4. Log into *zeus* from *apollo*. You might use Telnet or Secure Shell (SSH), for instance. The result should be the ability to type commands in a shell on *zeus*.
5. On *zeus*, type **export DISPLAY=apollo:0.0**. (This assumes you're using Bash; if you're using *tcsh*, the command would be **setenv DISPLAY apollo:0.0**.) This command tells *zeus* to use *apollo* for the display of X programs.
6. Type whatever you need to type to run programs at the *zeus* command prompt. For instance, you could type **ooffice** to launch OpenOffice.org. You should see the programs open on *apollo*'s display, but they're running on *zeus*—their computations use *zeus*'s CPU, they can read files accessible on *zeus*, and so on.
7. After you're done, close the programs you've launched, log off *zeus*, and type **xhost -zeus** on *apollo*. This will tighten security so that a miscreant on *zeus* cannot modify your display on *apollo*.

Sometimes, you can skip some of these steps. For instance, depending on how it's configured, SSH can forward X connections, meaning that SSH intercepts attempts to display X information and passes those requests on to the system that initiated the connection. When this happens, you can skip steps 3 and 5, as well as the *xhost* command in step 7. Using SSH to forward X requires support on the server, as described earlier in "Setting Up a Remote Access Server." A similar configuration on the client (using the *ForwardX11* option in */etc/ssh/ssh\_config*) is also required, or you can pass the *-X* option to *ssh* when making the initial connection. (Note that this is an uppercase *-X*; a lowercase *-x* disables X forwarding in the client!)

As an added security measure, many Linux distributions today configure X to ignore true network connections. If your distribution is so configured, the preceding steps won't work; when you try to launch an X program from the remote system, you'll get an error

message. To work around this problem, you must make an additional change, depending on how X is launched:

**GDM** On older versions of GDM, check the GDM configuration file (typically `/etc/X11/gdm/gdm.conf`): look for the line `DisallowTCP=true`, and change it to read `DisallowTCP=false`. On newer versions of GDM, edit `/etc/gdm/gdm.schemas`, and look for the line that reads `<key>security/DisallowTCP</key>`. A couple of lines below this, change the key from `true` to `false`.

**KDM or XDM** These two XDMCP servers both rely on settings in the `Xservers` file (in `/etc/X11/xdm` for XDM, and in this location or some other highly variable location for KDM). Look for the line that begins with `:0`. This line contains the command that KDM or XDM uses to launch the X server. If this line contains the string `-nolisten tcp`, remove that string from the line. Doing so eliminates the option that causes X to ignore conventional network connections.

**Special OpenSUSE configuration** In OpenSUSE, you must edit `/etc/sysconfig/displaymanager` and set the `DISPLAYMANAGER_XSERVER_TCP_PORT_6000_OPEN` option to `yes`.

**X launched from a text-mode login** If you log in using text mode and type **startx** to launch X, you may need to modify the `startx` script itself, which is usually stored in `/usr/bin`. Search this script for the string `-nolisten tcp`. Chances are this string will appear in a variable assignment (such as `defaultserverargs`) or possibly in a direct call to the X server program. Remove the `-nolisten tcp` option from this variable assignment or program call.

Once you've made these changes, you'll need to restart X. Thereafter, X should respond to remote access requests.



Distribution maintainers disable X's ability to respond to remote requests for a reason. If X responds to remote network requests, the risk of an intruder using a bug or misconfiguration to trick users by displaying bogus messages on the screen is greatly increased. Thus, you should disable this protection only if you're sure that doing so is necessary. You may be able to use an SSH link without disabling this protection.

Another option for running X programs remotely is to use the Virtual Network Computing (VNC) system (<http://www.realvnc.com>). VNC runs a special X server on the remote computer, and a special VNC client runs on the computer at which you sit. You use the client to directly contact the server. This reversal of client and server roles over the normal state of affairs with conventional X remote access is beneficial in some situations, such as when you are trying to access a distant system from behind certain types of firewall. VNC is also a cross-platform protocol; it's possible to control a Windows or Mac OS system from Linux using VNC, but this is not possible with X. (X servers for Windows and Mac OS are available, allowing you to control a Linux system from these non-Linux OSs.) Chapter 11 describes VNC in more detail.

## Remote GUI Logins

Chapter 1, “Getting Started with Linux,” described basic configuration of an X Display Manager Control Protocol (XDMCP) server, such as XDM, KDM, or GDM, to manage local GUI logins. These protocols also support remote GUI logins—you can use an extremely slim local system to access a more powerful remote computer. Some offices use this type of configuration, concentrating computing power in a small number of computers and enabling users to log in from old or dedicated hardware that might not be able to handle the needs of modern software. XDMCP-based logins differ from those initiated via a Telnet or SSH session in that you don’t need to first initiate that text-mode login. In an extreme case, you can sit down at a client and log in once, as if the XDMCP client system were the powerful remote computer.

### Configuring an XDMCP Server

To enable remote GUI access, you must make some changes to the standard XDMCP configuration described in Chapter 1. You make the first changes on the XDMCP server computer—that is, the system that will host user accounts, applications, and data files, as opposed to users’ desktop terminals.

The simplest case is XDM. With this server, you must first modify `/etc/X11/xdm/xdm-config`. Look for a line that resembles the following:

```
DisplayManager.requestPort: 0
```

This line tells XDM to not access a conventional server port. To use XDM for remote logins, you must change 0 to 177, the traditional XDMCP port.

A second change must be made to `/etc/X11/xdm/Xaccess`. This file controls what computers may access the XDM server, and in what ways. A wide-open system contains lines that use an asterisk (\*) to denote that anybody may access the system:

```
*
* CHOOSEER BROADCAST
```

This first line tells XDM that anybody may connect, and the second line tells XDM that anybody may request a *chooser*—a display of local systems that accept XDMCP connections. To limit the choices, you should list individual computers or groups of computers instead of using the asterisk wildcard:

```
*.pangaea.edu
tux.example.com
*.pangaea.edu CHOOSEER BROADCAST
```

This example permits any computer in the `pangaea.edu` domain to connect or receive a chooser, and it also lets `tux.example.com` connect but not receive a chooser.

The KDM server is configured much like XDM, but it may use different configuration files, as described in Chapter 1. Check `/etc/X11/kdm`, `/etc/kde/kdm`, or other locations suggested by a perusal of the contents of your KDM package. You may also need to look



for a file called `kdmrc`. Look for the `[Xdmcp]` section of this file and be sure the `Enable` option is set to `true`.

GDM completely abandons the XDM style of configuring remote logins. Instead, you should look for a file called `gdm.conf` or `custom.conf`, which is likely to be located in `/etc/X11/gdm` or `/etc/gdm`. Look for a section called `[xdmcp]` and be sure the `Enable` option is set to `true`.



Most Linux distributions have been taking extra steps in recent years to secure their default configurations. These steps may include setting firewall rules to block access to the XDMCP port, so if you have problems getting a newly configured XDMCP server to respond, you should check your firewall rules.

## Using an X Server

To use X to connect to a remote X server, you must alter the way X starts up. An easy way to test the configuration is to shut down the X server on the desktop system, log in using text mode, and then start X manually. A command such as the following should connect to a remote XDMCP server and permit a login:

```
$ Xorg -query zeus.example.com
```

This command starts X and tells it to query the specified computer for an XDMCP login server. If successful, the result should be a connection that looks very much like a local login, although it won't be as fast—all the display data must pass over the network after all.

In addition to `-query`, X supports two other XDMCP options: `-broadcast` and `-indirect server`. The `-broadcast` option starts an XDMCP broadcast, in which the XDMCP client (that is, the X server) sends a broadcast to the local network looking for XDMCP servers. The XDMCP client then either connects to the first server it finds or presents a list to the user. (In practice, Xorg-X11 and XFree86 seem to do the former, but some X servers do the latter.) The `-indirect server` option relies on the named server computer to present a list of XDMCP servers on the network.



Sometimes one of these three connection methods works when the others don't, so if you have problems, you should try all three methods.

If you want to permanently configure a computer as an *X terminal* (that is, a computer that functions only as an X server for remote computers, with few local programs running aside from the X server itself), you should locate whatever script starts X running on your system and change it. This script is likely to be a SysV startup script, as described in Chapter 4. Sometimes this script includes X options, such as `-no!isten tcp`, that interfere with X accepting remote connections, so you may need to remove such lines. (The preceding section, “Using X Programs Remotely,” provides additional information on such configurations.)

## Configuring Basic Printing

Printing in Linux is a cooperative effort involving several tools. A system administrator must be familiar with what each of the tools in this collection does, as well as how they interact. As with many other programs that are part of Linux, some of these tools have several versions, which can lead to confusion or incompatibilities if you're not aware of how the system as a whole functions. The basic Linux printing architecture is the same in all cases. One key component of this architecture is the presence of PostScript printers or the use of a program called Ghostscript to convert PostScript into a format that the printer can understand. Whether you use PostScript or non-PostScript printers, chances are your system uses the *Common Unix Printing System (CUPS)* utility to tie everything together. You should know how to configure CUPS, as well as how to use the utilities to submit and manage print jobs in Linux.

### The Linux Printing Architecture

Linux printing is built around the concept of a *print queue*. This is a sort of holding area where files wait to be printed. A single computer can support many distinct print queues. These frequently correspond to different physical printers, but you can also configure several queues to print in different ways to the same printer. For instance, you might use two queues to print using two different printer drivers, if each one has unique features or advantages.

Users submit print jobs by using a program called `lpr`. Users can call this program directly, or they may let another program call it. In either case, `lpr` sends the print job into a specified queue. This queue corresponds to a directory on the hard disk, typically in a subdirectory of the `/var/spool/cups` directory. The traditional Linux printing tool is called `lpd`, but all the major modern Linux distributions now use CUPS instead. This program runs in the background watching for print jobs to be submitted. The printing system accepts print jobs from `lpr` or from remote computers, monitors print queues, and serves as a sort of “traffic cop,” directing print jobs in an orderly fashion from print queues to printers.

One important and unusual characteristic of Linux printing is that it's highly network oriented. CUPS can accept print jobs that originate from remote systems as well as from local ones. In fact, even local print jobs are submitted via network protocols, although they don't normally use network hardware, so even a computer with no network connections can print. In addition to being a server for print jobs, CUPS can function as a client, passing print jobs on to other computers that run the same protocols.

The old `lpd` printing system is essentially unidirectional—applications know nothing about the printer's capabilities, so they blindly produce PostScript (as described shortly). The print queue takes this output and sends it on to the printer, which must deal with it as best it can. This is one of the deficiencies that CUPS corrected. Applications can query CUPS about a printer's capabilities—its paper sizes, whether it supports color, and so on. Support for these features is still far from universal, but it now exists in many Linux applications.

## Understanding PostScript and Ghostscript

If you've configured printers under Windows, Mac OS, OS/2, or certain other OSs, you're probably familiar with the concept of a *printer driver*. In these OSs, the printer driver stands between the application and the printer queue. In Linux, the printer driver is part of Ghostscript (<http://www.cs.wisc.edu/~ghost/>), which exists as part of the printer queue. This relationship can be confusing at times, particularly because not all applications or printers need Ghostscript. Ghostscript serves as a way to translate PostScript, a common printer language, into forms that can be understood by many different printers. Understanding Ghostscript's capabilities, and how it fits into a printer queue, can be important for configuring printers.

### PostScript: The De Facto Linux Printer Language

Most Unix (and therefore Linux) programs that print generate PostScript as an output format. In theory, any printer that understands PostScript can print the output of any program that generates PostScript. (In practice, there are various exceptions to this rule.)

A few programs don't generate PostScript output. Most commonly, many programs can produce raw text output. Such output seldom poses a major problem for modern printers, although some PostScript-only models choke on raw text. Some other programs can produce either PostScript or *Printer Control Language (PCL)* output for Hewlett-Packard laser printers or their many imitators. A very few programs can generate output that's directly accepted by other types of printers.

The problem with PostScript as a standard is that it's uncommon on the low- and mid-priced printers with which Linux is often paired. Therefore, to print to such printers using traditional Unix programs that generate PostScript output, you need a translator and a way to fit that translator into the print queue. This is where Ghostscript fits into the picture.

### Ghostscript: A PostScript Translator

Ghostscript is a PostScript interpreter that runs on a computer, offloading some of the need for RAM and CPU power from the printer to the computer. Ghostscript takes PostScript input and produces output in any of dozens of different bitmap formats, including formats that can be accepted by many non-PostScript printers. This makes Ghostscript a way to turn many inexpensive printers into Linux-compatible PostScript printers at very low cost. Ghostscript is available as open source software (GNU Ghostscript), with a more advanced variant (Aladdin Free Public License, or AFPL, Ghostscript) available for free. AFPL Ghostscript is not freely redistributable in any commercial package, though. Because all Linux distributions are available on CD-ROMs sold for a price, they ship with the older GNU Ghostscript, which works well enough for most users.

One of Ghostscript's drawbacks is that it produces large output files. A PostScript file that produces a page filled with text may be just a few kilobytes in size. If this page is to be printed on a 600 dots per inch (dpi) printer using Ghostscript, the resulting output file could be as large as 4MB—assuming it's black and white. If the page includes color, the size could be much larger. In some sense, this is unimportant because these big files will be

stored on your hard disk for only brief periods of time. They do still have to get from the computer to the printer, though, and this process can be slow. Also, some printers (particularly older laser printers) may require memory expansion to operate reliably under Linux.

## Squeezing Ghostscript into the Queue

Printing to a non-PostScript printer in Linux requires fitting Ghostscript into the print queue. This is generally done through the use of a *smart filter*. This is a program that's called as part of the printing process. The smart filter examines the file that's being printed, determines its type, and passes the file through one or more additional programs before the printing software sends it on to the printer. The smart filter can be configured to call Ghostscript with whatever parameters are appropriate to produce output for the queue's printer.

CUPS ships with a set of smart filters, which it calls automatically when you tell the system what model printer you're using. These filters are configured automatically as part of the CUPS configuration, as described in the upcoming section "Using the Web-Based CUPS Utilities."

The end result of a typical Linux printer queue configuration is the ability to treat any supported printer as if it were a PostScript printer. Applications that produce PostScript output can print directly to the queue. The smart filter detects that the output is PostScript and runs it through Ghostscript. The smart filter can also detect other file types, such as plain text and various graphics files, and it can send them through appropriate programs, instead of or in addition to Ghostscript, in order to create a reasonable printout.

If you have a printer that can process PostScript itself, the smart filter is usually still involved, but it doesn't pass PostScript through Ghostscript. In this case, the smart filter passes PostScript directly to the printer, but it still sends other file types through whatever processing is necessary to turn them into PostScript.

## Running a Printing System

Because Linux printing systems run as daemons, they must be started before they're useful. This task is normally handled automatically via startup scripts in `/etc/rc.d` or `/etc/rc?.d` (where ? is a runlevel number). Look for startup scripts that contain the string `cups` in their names (or `lpd` or `lprng` for the older BSD LPD or LPRng printing systems). If you're unsure if a printing system is currently active, use the `ps` utility to search for running processes by these names, as follows:

```
$ ps ax | grep cups
3713 ? S 0:00 cupsd
```



**NOTE**

The `ps` command is covered in more detail in Chapter 3. The `grep` command and pipes (used to link these two commands together) are covered in Chapter 2.



## Real World Scenario

### Choosing an Appropriate Printer for Linux

If you want a speedy printer for Linux, choose a model with built-in PostScript. In my experience, Ghostscript-driven printers work well enough for 600 dpi black-and-white printers with speeds of up to about 6 pages per minute (ppm), although theoretically both the parallel port and USB 1.x port should be able to handle speeds of 3–5 times that value. If the printer's speed is greater than that, the parallel or USB 1.x port may not be able to deliver the necessary performance, although you may be able to tweak it to get somewhat better speed. Non-PostScript printers that use USB 2.0 can handle higher resolutions and color. Modern office printers often claim speeds of 30–40 ppm. To improve your odds of attaining that speed, ensure that they have USB 2.0 or network ports and understand PostScript natively.

Color inkjet printers are generally limited more by the speed of the print head than by the speed of the data coming over their ports. Few such printers directly support PostScript. The Ghostscript support for these models varies from nonexistent to excellent. Some models come with Windows-based PostScript engines that are conceptually similar to Ghostscript, but such software is useless under Linux. There are a few color PostScript printers that use non-inkjet printing technologies. Color laser printers, in particular, have become popular in recent years.

For information on what printers are supported by Ghostscript, check the Ghostscript Web page or the OpenPrinting Web page ([http://www.linuxprinting.org/printer\\_list.cgi](http://www.linuxprinting.org/printer_list.cgi)).

This example shows that `cupsd`, the CUPS daemon, is running, so the system is using CUPS for printing. If you can't find any running printing system, consult your distribution's documentation to learn what is available and check that the appropriate package is installed. All major distributions include startup scripts that should start the appropriate printing daemon when the computer boots.

## Configuring CUPS

CUPS uses various configuration files in the `/etc/cups` directory and its subdirectories. You can edit these files directly and may need to do so if you want to share printers or use printers shared by other CUPS systems. The simplest way to add printers to CUPS, though, is to use the tool's Web-based configuration utility.

### Editing the CUPS Configuration Files

You can add or delete printers by editing the `/etc/cups/printers.conf` file, which consists of printer definitions. Each definition begins with the name of a printer, identified by

the string `DefaultPrinter` (for the default printer) or `Printer` (for a nondefault printer) in angle brackets (<>), as in the following:

```
<DefaultPrinter okidata>
```

This line marks the beginning of a definition for a printer queue called `okidata`. The end of this definition is a line that reads `</Printer>`. Intervening lines set assorted printer options, such as identifying strings, the printer's location (its local hardware port or network location), its current status, and so on. Additional options are stored in a *PostScript Printer Definition* (PPD) file that's named after the queue and stored in the `/etc/cups/ppd` subdirectory. PPD files follow an industry-standard format. For PostScript printers, you can obtain a PPD file from the printer manufacturer, typically from a driver CD-ROM or from the manufacturer's Web site. CUPS and its add-on driver packs also ship with a large number of PPD files that are installed automatically when you use the Web-based configuration utilities.

As a general rule, you're better off using the CUPS Web-based configuration tools to add printers, rather than adding printers by directly editing the configuration files. If you like, though, you can study the underlying files and tweak the configurations using a text editor to avoid having to go through the full Web-based tool to make a minor change.

One exception to this rule relates to configuring the CUPS Web-based interface tool itself and CUPS's ability to interface with other CUPS systems. One of the great advantages of CUPS is that it uses a network printing protocol, known as the *Internet Printing Protocol* (IPP), that supports a feature it calls *browsing*. This feature enables computers on a network to automatically exchange printer lists, which can greatly simplify configuring network printing. You may need to change some settings in the main CUPS configuration file, `/etc/cups/cupsd.conf`, to enable this support.



The older BSD LPD and LPRng tools used a different network printing protocol. CUPS can print to network printers using this protocol, and you can run the `cups-lpd` server to accept LPD jobs from older computers on a system that runs CUPS.

The `/etc/cups/cupsd.conf` file contains a number of configuration blocks that specify which other systems should be able to access it. Each block controls access to a particular location on the server. These blocks look like this:

```
<Location /printers>
Order Deny,Allow
Deny from All
BrowseAllow from 127.0.0.1
BrowseAllow from 192.168.1.0/24
BrowseAllow from @LOCAL
Allow from 127.0.0.1
Allow from 192.168.1.0/24
Allow from @LOCAL
</Location>
```



If you're configuring a workstation with a local printer that you don't want to share or if you want to configure a workstation to use printers shared via LPD or some other non-IPP printing protocol, you shouldn't need to adjust `/etc/cups/cupsd.conf`. If you want to access remote IPP printers, however, you should at least activate browsing by setting the directive `Browsing On`, as described shortly. You shouldn't have to modify your location definitions unless you want to share your local printers.

The `/printers` location, shown here, controls access to the printers themselves. Features of this example include the following:

**Directive order** The `Order Deny,Allow` line tells CUPS in which order it should apply allow and deny directives—in this case, allow directives modify deny directives.

**Default policy** The `Deny from All` line tells the system to refuse all connections except those that are explicitly permitted.

**Browsing control lines** The `BrowseAllow` lines tell CUPS from which other systems it should accept browsing requests. In this case, it accepts connections from itself (127.0.0.1), from systems on the 192.168.1.0/24 network, and from systems connected to local subnets (@LOCAL).

**Access control lines** The `Allow` lines give the specified systems nonbrowse access to printers—that is, those systems can print to local printers. In most cases, the `Allow` lines will be the same as the `BrowseAllow` lines.

You can also create a definition that uses `Allow from All` and then creates `BrowseDeny` and `Deny` lines to limit access. As a general rule, though, the approach shown in the preceding example is safer. Locations other than the `/printers` location can also be important. For instance, there's a `root (/)` location that specifies default access permissions to all other locations and an `/admin` location that controls access to CUPS administrative functions.

Before the location definitions in `cupsd.conf` are a few parameters that enable or disable browsing and other network operations. You should look for the following options specifically:

**Enabling browsing** The `Browsing` directive accepts `On` and `Off` values. The CUPS default is to enable browsing (`Browsing On`), but some Linux distributions disable it by default.

**Browsing access control** The `BrowseAddress` directive specifies the broadcast address to which browsing information should be sent. For instance, to broadcast data on your printers to the 192.168.1.0/24 subnet, you'd specify `BrowseAddress 192.168.1.255`.

Once you've configured a CUPS server to give other systems access to its printers via appropriate location directions, and once you've configured the client systems to use browsing via `Browsing On`, all the systems on the network should autodetect all the printers on the network. There's no need to configure the printer on any computer except the one to which it's directly connected. All printer characteristics, including their network locations and PPD files, are propagated automatically by CUPS. This feature is most important in configuring large networks with many printers or networks on which printers are frequently added and deleted.



## Obtaining CUPS Printer Definitions

The basic version of CUPS ships with smart filter support for just a few printers, including raw queues that do no processing and a few models from Hewlett-Packard, Epson, and Okidata. If you use another printer, you should obtain extra CUPS printer definitions. These definitions may consist of PPD files, appropriate behind-the-scenes “glue” to tell CUPS how to use them, and possibly Ghostscript driver files. These printer definitions can be obtained from several sources:

**Your Linux distribution** Many distributions ship extra printer definitions under various names, so check your distribution for such a package. Many distributions provide one of the driver packages that’s described next.

**Foomatic** The OpenPrinting Web site hosts a set of utilities and printer definitions known collectively as Foomatic (<http://www.linuxfoundation.org/en/OpenPrinting/Database/Foomatic>). These provide many additional printer definitions for CUPS (as well as for other printing systems).

**Gutenprint** These drivers were originally known as GIMP Print, after the GNU Image Manipulation Program (GIMP) from which they originated. These drivers support a wide variety of printers. Check <http://gimp-print.sourceforge.net> for more information.

**CUPS DDK** The CUPS Driver Development Kit (DDK; <http://www.cups.org/software.php>) is a set of tools designed to simplify CUPS driver development. It ships with a handful of sample drivers.

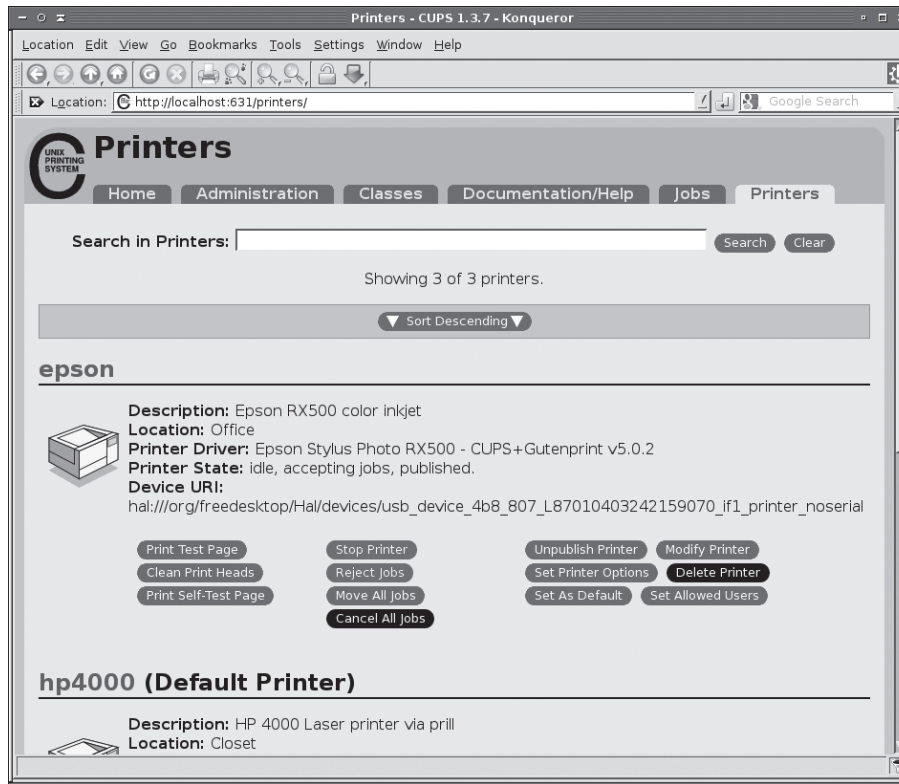
**Printer manufacturers** Some printer manufacturers offer CUPS drivers for their printers. These may be nothing more than the Foomatic, Gutenprint, or other open source drivers; but a few provide proprietary drivers, some of which support advanced features that the open source drivers don’t support.

Chances are good that you’ll find your printer, or something close enough to work well, in your distribution’s standard set of drivers. If you try to configure your printer and can’t find a suitable driver in the CUPS Web-based tools, though, you should check with these sources for additional drivers. You might also try using your Internet-enabled package management tools (such as `yum` or `apt-get`) to locate drivers. Try searching the package database for packages that include `cups` or `print` in their names.

## Using the Web-Based CUPS Utilities

The CUPS IPP printing system is closely related to the Hypertext Transfer Protocol (HTTP) used on the Web. The protocol is so similar, in fact, that you can access a CUPS daemon by using a Web browser. You need only specify that you want to access the server on port 631—the normal printer port. To do so, enter **`http://localhost:631`** in a Web browser on the computer running CUPS. (You may be able to substitute the hostname, or access CUPS from another computer by using the other computer’s hostname, depending on your `cupsd.conf` settings.) This action brings up a list of administrative tasks you can perform. Click **Printers** to open the printer management page, as shown in Figure 9.3. Clicking **Administration** brings up a page in which you can set various administrative options, including adding new printers.



**FIGURE 9.3** CUPS provides its own Web-based configuration tool.**NOTE**

If you're configuring a stand-alone computer or the only one on a network to use CUPS, the printer list will be empty, unlike the one shown in Figure 9.3. If other computers on your network use CUPS, you may see their printers in the printer list, depending on their security settings.

You can add, delete, or modify printer queues using the CUPS Web control system. To add a printer, follow these steps:

1. From the administration page, click Add Printer.
2. The system displays a page asking for the printer's name, location, and description. Enter appropriate information in the Name, Location, and Description fields. These fields are all entirely descriptive, so enter anything you like. (Users will use your entry in the Name field to access the printer, though.) When you click Continue, CUPS asks for the printer device.

3. The printer device may be a local hardware port (such as a parallel printer port or a USB port), a remote LPD printer, a remote SMB/CIFS (Samba) printer, or other devices. The precise options available vary from one distribution to another. Select the appropriate one from the pop-up list, and click Continue.
4. If you entered a network printer, the result is a page in which you enter the complete path to the device. Type the path, such as **lpd://printserv/brother** to print to the brother queue on the printserv computer. Click Continue when you're done.
5. If you entered a local device in step 3 or after you've entered the complete path in step 4, you'll see a list of driver classes, such as PostScript and HP. Select one, and click Continue. Alternatively, you can point directly to a PPD file if you have one handy. If you do this, you'll skip the next step.
6. CUPS now displays a complete list of printer models within the class you selected in step 5. Select an appropriate model, and click Add Printer. Alternatively, you can point directly to a PPD file.
7. If you haven't already performed administrative actions with CUPS, it will ask you for a username and password. Type **root** as the username and the administrative password as the password, and then click OK.



CUPS doesn't encrypt its data, so you shouldn't use it to administer printers remotely. Doing so would be a security risk, because the passwords would be exposed to sniffing. If necessary, you can use an SSH tunnel to overcome this limitation.

8. CUPS informs you that the printer has been added.
9. If you wait a few seconds, the notification that the printer has been added is replaced by a page in which you can set printer-specific options, such as sheet feeder sources and print resolutions. Adjust any options you like, and then click Set Printer Options.

If you click the Printers item at the top of the page, you should be returned to the printers list (see Figure 9.3), but your new printer should be listed among the existing queues. You can print a test page by clicking Print Test Page. If all goes well, a test page will emerge from your printer. If it doesn't, go back and review your configuration by clicking Modify Printer. This action takes you through the steps for adding a printer but with your previous selections already entered as the defaults. Try changing some settings until you get the printer to work.

From the printer queue list, you can also click Set Printer Options to set various printer options. What options are available depends on the printer, but common options include the resolution, color dithering options, the paper size, whether or not to enable double-sided printing, and the presence of banner pages.

Many distributions provide dedicated GUI tools for printer configuration, such as Printer Configuration (aka `system-config-printer`). These tools usually mirror the standard Web-based CUPS configuration tool in features. Using them is conceptually similar to the procedure just described, although details differ—you may need to locate options on menus or enter data in slightly different ways.

## Printing to Network Printers

If your network hosts many Windows computers, you may use the Server Message Block/Common Internet File System (SMB/CIFS) for file and printer sharing among Windows systems. Linux's Samba server also implements this protocol and so can be used for sharing printers from Linux.



Chapter 11 describes the basics of configuring a Linux Samba server, so consult it if you want to share an existing printer queue with Windows clients.

On the flip side, you can print to an SMB/CIFS printer queue from a Linux system. To do so, you select an SMB/CIFS queue in the CUPS printer configuration. Under CUPS, this option is called Windows Printer via SAMBA in step 3 of the procedure in the preceding section. You must then provide your username, password, server name, and share name, but the format is not obvious from the Web-based configuration tool:

```
smb://username:password@SERVER/SHARE
```

This is a uniform resource identifier (URI) for an SMB/CIFS share. You must substitute appropriate values for *username*, *password*, *SERVER*, and *SHARE*, of course. Once this is done and you've finished the configuration, you should be able to submit print jobs to the SMB/CIFS share.



SMB/CIFS printers hosted by Windows systems are usually non-PostScript models, so you must select a local Linux printer definition, just as you would for a local printer. Printers hosted by Linux systems running Samba, though, are frequently configured to act like PostScript printers, so you should select a PostScript driver when connecting to them.

If you want to print to a Unix or Linux server that uses the old LPD protocol, the URI format is similar but omits a username and password:

```
lpd://hostname/queue
```

You can use the same format, but substitute `ipp://` for `lpd://`, to print to a CUPS server if browsing is disabled on your network.

In practice, you may be faced with a decision: should you use LPD, IPP, or SMB/CIFS for submitting print jobs? To be sure, not all print servers support all three protocols, but a Linux server might support them all. As a general rule, IPP is the simplest to configure because it supports browsing, which means that CUPS clients shouldn't need explicit configuration to handle specific printers. This makes IPP the best choice for Linux-to-Linux printing, assuming both systems run CUPS. When CUPS isn't in use, LPD is generally easier to configure than SMB/CIFS, and it has the advantage of not requiring a username or password to control access. Because SMB/CIFS security is password-oriented, clients

typically store passwords in an unencrypted form on the hard disk. This fact can become a security liability, particularly if you use the same account for printing as for other tasks. On the other hand, sometimes use of a password on the server provides more of a security benefit than the risk of storing that password on the client. Generally speaking, if clients are few and well protected, while the server is exposed to the Internet at large, using passwords can be beneficial. If clients are numerous and exposed to the Internet while the print server is well protected, though, a password-free security system that relies on IP addresses may be preferable.

## Monitoring and Controlling Print Queues

You can use several utilities to submit print jobs and to examine and manipulate a Linux print queue. These utilities are `lp`, `lpr`, `lpq`, `lprm`, `cancel`, `lpc`, and `lpstat`. All of these commands can take the `-P` parameter to specify that they operate on a specific print queue. In addition, CUPS' Web-based interface permits point-and-click management of print queues.

## Printing Files

Once you've configured the system to print, you probably want to start printing. As mentioned earlier, Linux uses the `lpr` program to submit print jobs. Alternatively, you may use `lp`. These programs accept many options for modifying the programs' actions. The more common options are summarized in Table 9.3, but you may want to check your preferred utility's man page for additional options. The reason for two similar programs is historical; they originated on two different Unix varieties, and because CUPS serves as a substitute for both, it provides workalikes for both utilities.

**TABLE 9.3** `lpr` and `lp` Options and Effects

<code>lpr</code> Option	<code>lp</code> Option	Effect
<code>-P queueName</code>	<code>-d queueName</code>	This option enables you to specify a print queue. This is useful if you have several printers or if you've defined several queues for one printer. If you omit this option, the default printer is used. (The original BSD version of <code>lpr</code> required that no space be present between <code>-P</code> and the queue name, but CUPS doesn't suffer from this limitation.)
<code>-r</code>	None	Normally, <code>lpr</code> sends a copy of the file you print into the queue, leaving the original unharmed. Specifying the <code>-r</code> option causes <code>lpr</code> to delete the original file after printing it.
<code>-E</code>	<code>-E</code>	This option enables encryption on the connection to the print server.

TABLE 9.3 lpr and lp Options and Effects (continued)

lpr Option	lp Option	Effect
-C <i>jobname</i> , -J <i>jobname</i> , or -T <i>jobname</i>	-t " <i>jobname</i> "	Print jobs have names to help identify them, both while they're in the queue and once printed (if the queue is configured to print banner pages). The name is normally the name of the first file in the print job, but you can change it by including this option.
-# <i>number</i>	-n <i>copies</i>	You can specify the number of copies of a print job by including this option.
-l	None	This option causes CUPS to bypass its smart filter for the printer.
-m	-m	You can have CUPS send you an e-mail when the job has finished by using this option.
None	-P <i>page-list</i>	Sets a range of pages to be printed. Pages may be listed individually, separated by commas (as in 1, 7, 9), be a range separated by a dash (as in 2-4), or use both methods (as in 1, 7-9, 12).

Suppose you have a file called `report.txt` that you want to print to the printer attached to the `lexmark` queue. This queue is often quite busy, so you want the system to send e-mail to your account when it's done so that you know when to pick up the printout. You could use the following command to accomplish this task:

```
$ lpr -Plexmark -m report.txt
```

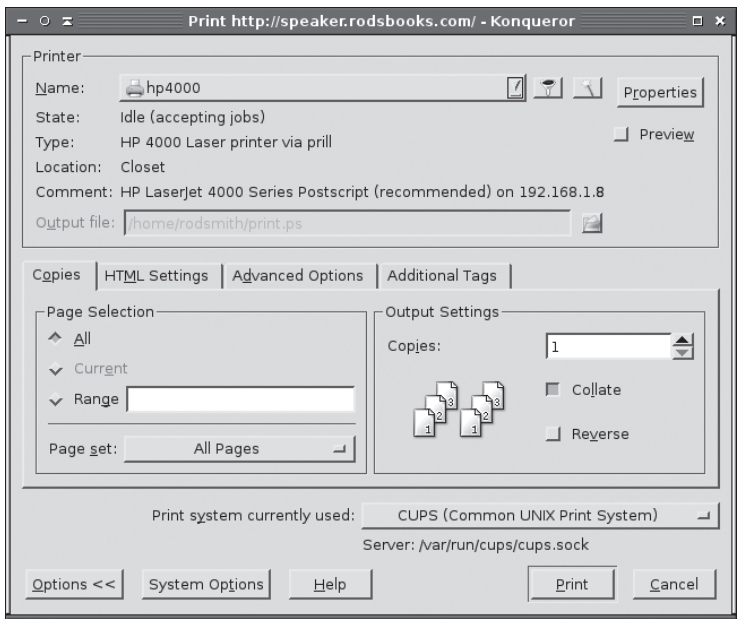
The equivalent command using `lp` is as follows:

```
$ lp -d lexmark -m report.txt
```

The `lpr` and `lp` commands are accessible to ordinary users as well as to `root`, so anybody may print using these commands.

Many programs that need to print directly, such as graphics programs and word processors, call `lpr` or `lp`. Other programs interface more directly with CUPS. In either case, these programs typically give you some way to adjust the print command so that you can change parameters such as the printer name. For instance, Figure 9.4 shows Konqueror's Print dialog box. Konqueror features a pop-up list button that lets you select the print queue. This is the Name field in Figure 9.4, but some programs call it something else. Some programs also provide a text entry field in which you type some or all of an `lpr` or `lp` command, instead of selecting from a pop-up list of available queues. Consult the program's documentation if you're not sure how it works.

**FIGURE 9.4** Most X-based programs that print enable you to set printer options via a dialog box.



## Displaying Print Queue Information

The `lpq` utility displays information on the print queue—how many files it contains, how large they are, who their owners are, and so on. By entering the user's name as an argument, you can also use this command to check on any print jobs owned by a particular user. To use `lpq` to examine a queue, you might issue a command like the following:

**\$ `lpq -Pepson`**

epson is ready and printing

Rank	Owner	Job	File(s)	Total Size
active	rodsmi	323	reply.tif	884736 bytes



This example shows the `lpq` output for CUPS. If you use another printing system, the format will differ, although similar information should appear.

Of particular interest is the job number—323 in this example. You can use this number to delete a job from the queue or reorder it so that it prints before other jobs. Any user may use the `lpq` command.

Instead of using `lpq`, you may use `lpstat`. This utility's purpose is similar, but its details differ. Most notably, it takes a lowercase `-p` to specify the printer name, and its output is formatted differently:

```
$ lpstat -p epson
```

```
printer epson now printing epson-323. enabled since Sat 09 May 2009 10:14:35 AM EDT
```

```
Gutenprint Printing page 1, 12%
```

The default output of `lpstat` is less complete than that of `lpq`. The job ID number is present, but it's not labeled as such: note the string `epson-323` in the first line of output. The number (323) is the job ID number.

## Removing Print Jobs from the Queue

The `lprm` or `cancel` command removes one or more jobs from the print queue. There are several ways to issue this command:

- If it's issued with a number, that number is understood to be the job ID (as shown in `lpq`'s output) that's to be deleted.
- If root runs the BSD `lprm` and passes a dash (-) to the program, it removes all print jobs belonging to all users.

The `lprm` and `cancel` programs may be run by root or by an ordinary user, but as just noted, their capabilities vary depending on who runs them. Ordinary users may normally remove only their own jobs from the queue, but root may remove anybody's print jobs. (CUPS does provide an option, which can be set from the Web interface, to enable ordinary users to cancel other users' jobs, but this option is not set by default.)

Instead of using `lprm` or `cancel`, you may use the Web interface for CUPS to remove jobs from a print queue. Click the Jobs tab (shown in Figure 9.3) to obtain a list of current print jobs. Click Cancel Job in the row for any particular job to cancel it.

## Controlling the Print Queue

In the original BSD LPD system, the `lpc` utility starts, stops, and reorders jobs within print queues. Although CUPS provides an `lpc` command, it has few features. Instead of using `lpc`, you should use the CUPS Web interface, which provides point-and-click print queue management:

- You can disable a queue by clicking the Stop Printer link for the printer on the CUPS Web interface (see Figure 9.3). When you do so, this link changes to read Start Printer, which reverses the effect when clicked. The Jobs tab also provides a way to cancel and otherwise manage specific jobs.
- You can use a series of commands, such as `cupsenable`, `cupsdisable`, and `lpmove`, to control the queue. These commands enable a queue, disable a queue, or move a job from one queue to another. Moving a job can be handy if you must shut down a queue for maintenance and want to redirect the queue's existing jobs to another printer.

## Summary

Beyond basic network configuration lie increasingly advanced network configuration tasks. Router and firewall configurations are two such sets of tasks. You can enable a computer with two or more network interfaces to pass data between network segments, thus linking them, either on a multisegment private network or to link one network to the Internet. Firewall configurations can improve the security of router configurations, but firewalls are also important security tools for nonrouter computers. In Linux, you use the `iptables` program to create firewalls.

Two types of server programs that you're likely to run on Linux are remote login tools and printing tools. Basic remote login tools permit text-mode access to the computer. Although Telnet was historically popular for this task, today SSH has largely supplanted Telnet because of SSH's superior security features. In addition to supporting encryption, SSH supports file transfers and tunneling a wide variety of protocols, which enables the addition of encryption features to otherwise unencrypted protocols. This feature is frequently used in conjunction with X, so users can log into a Linux system remotely and run X-based programs without taking special steps. You can also use X remotely without tunneling it through SSH, but this requires extra steps to link the client and servers. In some environments, you may want to reconfigure your X and XDMCP servers to permit remote GUI logins.

Printing in Linux is a network task; even on an isolated workstation, the CUPS print queue is a network-enabled tool, and it uses network protocols locally to do its work, much like X on an isolated system. CUPS really shines in a networked environment, though; it supports automatic printer discovery so that all the printers on a network may be shared by all the computers, with no need to explicitly configure these printers except on the computer that most directly manages them. Command-line tools such as `lpr`, `lprm`, `lpq`, and `lpstat` enable you to submit and manage print jobs; or you can use CUPS's Web-based tools to manage the task.

## Exam Essentials

**Explain the function of a packet-filter firewall.** A packet-filter firewall filters network traffic based on low-level data that's easily obtained from individual network packets without reassembling them into a complete data stream. Examples include source and destination IP addresses and port numbers. Such filters can protect a computer or (when used on a router) a network from certain types of suspicious network traffic, such as off-site attempts to access sensitive servers intended for local use.

**Summarize how a packet-filter firewall is created in Linux.** The usual tool for creating a packet-filter firewall is `iptables`. Each call to this utility creates an individual rule, such as one to drop incoming packets to a particular port from certain IP addresses. Many



such rules combine together to create a complete firewall. Typically, a script runs at system startup to manage this task.

**Describe common text-mode remote login tools.** Telnet was the traditional remote text-mode login tool, but it passes data over the network unencrypted and so is a poor choice today. The Secure Shell (SSH) is the most common remote text-mode access tool today.

**Explain the nature of X clients and servers.** An X server controls a screen display and handles input from the user's mouse and keyboard. Therefore, the X server is used directly by the user, and X clients are the programs that rely on the X server's services.

**Explain the role of an XDMCP server.** An XDMCP server, such as XDM, KDM, or GDM, launches X and controls access to X via a login prompt—that is, it serves as Linux's GUI login system. XDMCP servers are also network enabled, providing a way to log in remotely from another X server.

**Explain the role of Ghostscript in Linux printing.** PostScript is the standard Linux printing language, and Ghostscript converts PostScript into bitmap formats that are acceptable to non-PostScript printers. Thus, Ghostscript is a critical translation step in most Linux print queues.

**Summarize how print jobs are submitted and managed under Linux.** You use `lpr` or `lp` to submit a print job for printing, or an application program may call `lpr` or `lp` itself or implement its functionality directly. The `lpq` or `lpstat` utility summarizes jobs in a queue, and `lprm` can remove print jobs from a queue.

## Review Questions

1. What is the default port used by the Simple Mail Transfer Protocol (SMTP)?
  - A. 143
  - B. 80
  - C. 25
  - D. 21
2. Which of the following ports are known as *unprivileged* ports?
  - A. Those that have numbers above 1024
  - B. Those that have numbers between 512 and 1024
  - C. Those that have numbers between 1 and 100
  - D. Those that have numbers below 1024
3. Which port, by default, is commonly used by OpenSSH?
  - A. 20
  - B. 21
  - C. 22
  - D. 23
4. Which port, by default, is commonly used by HTTPS?
  - A. 111
  - B. 143
  - C. 389
  - D. 443
5. Which of the following is true of the DROP policy for iptables?
  - A. It replies as if the computer was available but running no software on the target port.
  - B. It ignores packets, providing the illusion of a network error between the sender and recipient.
  - C. It may be used only on ports that are opened by servers, not clients.
  - D. It may be applied as a default policy but not on a port-by-port basis.

6. When is it reasonable to enable routing between network segments on a computer that's connected to both of them?
  - A. Whenever one of the network segments is connected to the Internet and the other segment is a private network.
  - B. Whenever the segments must communicate directly and the computer in question is configured as a firewall.
  - C. Whenever the segments must communicate directly and the computer in question is designated as the router.
  - D. Never; this task should be handled only by dedicated network hardware that never runs Linux.
7. A firewall script includes the following two lines. What is their purpose?
 

```
iptables -A OUTPUT -d 127.0.0.1 -o lo -j ACCEPT
iptables -A INPUT -s 127.0.0.1 -i lo -j ACCEPT
```

  - A. To set the default policy for all chains to ACCEPT; subsequent rules will use DROP or REJECT.
  - B. To enable routing of localhost traffic on a computer configured as a router.
  - C. To enable communications over the localhost interface for local programs.
  - D. To set the default policy for the OUTPUT and INPUT chains to ACCEPT, leaving the FORWARD chain unaffected.
8. Broadly speaking, how will use of `iptables` on a router with firewall features differ from its use on a workstation?
  - A. A router's `iptables` rules will include an emphasis on the FORWARD chain; a workstation's will emphasize the INPUT and OUTPUT chains.
  - B. A router's `iptables` rules will most likely use a default DROP policy, whereas a workstation's will probably use a default ACCEPT policy.
  - C. A router's `iptables` rules will be activated by a script, whereas a workstation's will be configured using a GUI tool.
  - D. A router's `iptables` rules will emphasize privileged port numbers, whereas a workstation's will emphasize unprivileged port numbers.
9. You want to use an X server on an old Pentium computer to run X clients on a modern Itanium CPU system, with the goal of performing computationally intensive spreadsheet calculations. Which of the following is true?
  - A. The spreadsheet will compute slowly because of the slow speed of the Pentium server.
  - B. You won't be able to run the spreadsheet because the Itanium and Pentium CPUs need different executables.
  - C. The computation will run swiftly, but graphics displays may be slowed by the Pentium's limited speed.
  - D. Computations will run swiftly only if the Itanium computer makes its filesystem available via NFS.

10. Why is it unwise to allow `root` to log on directly using SSH?
  - A. Somebody with the `root` password but no other password could then break into the computer.
  - B. The `root` password should never be sent over a network connection; allowing `root` logins in this way is inviting disaster.
  - C. SSH stores all login information, including passwords, in a publicly readable file.
  - D. When logged on using SSH, `root`'s commands can be easily intercepted and duplicated by undesirable elements.
11. What programs directly support secure encrypted file transfers via SSH? (Choose all that apply.)
  - A. `ftp`
  - B. `sftp`
  - C. `scp`
  - D. NFS
12. What program can you use to generate a key that will enable logins to remote SSH servers without using a password?
  - A. `authorized_keys`
  - B. `ssh-keygen`
  - C. `sshpasswd`
  - D. `id_rsa`
13. To whom should you distribute your server's main SSH private key?
  - A. Only to servers with which you want to communicate
  - B. Only to servers that have already provided their own public keys
  - C. To anybody who can provide a matching SSH public key
  - D. None of the above
14. Which of the following is generally true of Linux programs that print?
  - A. They send data directly to the printer port.
  - B. They produce PostScript output for printing.
  - C. They include extensive collections of printer drivers.
  - D. They can print only with the help of add-on commercial programs.
15. Which of the following describes the function of a smart filter?
  - A. It detects the type of a file and passes it through programs to make it printable on a given model of printer.
  - B. It detects information in print jobs that might be confidential, as a measure against industrial espionage.
  - C. It sends e-mail to the person who submitted the print job, obviating the need to wait around the printer for a printout.
  - D. It detects and deletes prank print jobs that are likely to have been created by miscreants trying to waste your paper and ink.

16. What information about print jobs does the `lpq` command display? (Choose all that apply.)
- A. The name of the application that submitted the job
  - B. A numerical job ID that can be used to manipulate the job
  - C. The amount of ink or toner left in the printer
  - D. The username of the person who submitted the job
17. You try to create a definition for a new printer using the CUPS Web-based tool, but you can't find a suitable printer definition. How might you proceed?
- A. Edit the `/etc/cups/printers.conf` file directly.
  - B. Use the `system-config-printer` program instead of the CUPS Web tool.
  - C. Install the LPRng package, if it's not already installed.
  - D. Install the Foomatic package, if it's not already installed.
18. In setting up a network printer for use in Linux, a system administrator uses the following URI:
- ```
smb://bsd:lpd@USB/PARALLEL
```
- Assuming this URI is correct, what type of printer is the administrator attempting to use?
- A. A local parallel-port printer
 - B. A local USB printer
 - C. A networked Windows or Samba printer
 - D. A networked BSD LPD printer
19. You type `lpr -P brother sample.ps` file to submit a print job. What would you type to do the same thing using `lp`?
- A. `lp -d brother sample.ps`
 - B. `lp -p brother sample.ps`
 - C. `lp --printer brother sample.ps`
 - D. `lp -P brother sample.ps`
20. Under what circumstances may ordinary users cancel other users' print jobs on a system that uses CUPS?
- A. Only if the print queue has registered an error condition (paper out, printer jam, and so on)
 - B. Only if the administrator has enabled the option to allow this activity
 - C. Only if permissions on the original file or its directory permit other users to delete the file
 - D. Never

Answers to Review Questions

1. C. The SMTP service by default uses port 25. Port 143 is used by IMAP, port 80 is used for WWW, and port 21 is used by FTP.
2. A. *Unprivileged* ports are those that have numbers above 1024. *Privileged* ports are those that have numbers below 1024. The idea is that a client can connect to a privileged port and be confident that the server running on that port was configured by the system administrator.
3. C. The default port for OpenSSH is 22. FTP uses ports 20 and 21, while Telnet uses port 23.
4. D. The default port for HTTPS is 443. PortMapper uses port 111, while the default port for IMAP 2 is 143. The default port for LDAP is 389.
5. B. The DROP policy causes the computer to respond as described in option B. Option A describes the REJECT policy, not the DROP policy. Contrary to option C, DROP may be used on any port, whether it's used by clients or servers. Contrary to option D, DROP may be used either as a default policy or on a port-by-port basis.
6. C. Option C describes the basic reason for enabling routing in a computer. Option A is, in some sense, backwards; routing is not normally enabled between an isolated private network and the Internet. Option B is incorrect because a firewall configuration is not required in a router, although it's often advisable. Both options A and B also ignore the fact that another computer might already be configured as a router; two networks might share multiple computers, but only one needs to be configured as a router to enable communication between them. Contrary to option D, both dedicated router devices and general-purpose computers may function as routers.
7. C. Linux uses the localhost (127.0.0.1) IP address for local communications; network-enabled programs use this address for communications even on one system. If your default `iptables` policy is DROP or REJECT, this interface will be blocked, so it's necessary to unblock it using lines like those shown in the question, making option C correct. The default policy, referenced in options A and D, is set using the `-P` option, as in `iptables -t filter -P FORWARD DROP`, so these options are incorrect. Routing of localhost traffic makes no logical sense—localhost traffic is, by definition, local and therefore is not routed—so option B is incorrect.
8. A. The FORWARD chain is used for forwarding traffic, as in a router, so a router's firewall will include rules that affect the way the router forwards network traffic. A workstation doesn't forward traffic, so there will be few or no rules affecting the FORWARD chain on a workstation, making option A correct. Contrary to option B, both workstations and routers may use either the DROP policy or the ACCEPT policy (or the REJECT policy) as a default, although ACCEPT is the least-preferred policy. Although both scripts and GUI tools can be used to create a firewall, there is no necessary link between these tools and the type of computer, as option C suggests, so that option is incorrect. Option D is incorrect because, although the distinction between privileged and unprivileged port numbers is an important one, only the server's port numbers (which are, by and large, privileged) are fixed, so even a workstation's `iptables` rules will emphasize these port numbers.

9. C. The X server handles the display and user input only, so its speed will influence graphics displays. Computations occur on the fast Itanium-based X client system.
10. A. Allowing only normal users to log in via SSH effectively requires two passwords for any remote `root` maintenance, improving security. SSH encrypts all connections, so it's unlikely that the password, or commands issued during an SSH session, will be intercepted, which make options B and D incorrect. (Nonetheless, some administrators prefer not to take even this small risk.) SSH doesn't store passwords in a file, making option C incorrect.
11. B, C. The `sftp` program provides an `ftp`-like user interface that works via SSH, while `scp` is an SSH-enabled program that works like `cp`. Both enable you to transfer files using encryption via SSH. (`sftp` requires explicit support in the SSH server, though.) The `ftp` program supports unencrypted transfers, and not via SSH, so option A is incorrect. The Network File System (NFS) is another unencrypted protocol, so option D is also incorrect.
12. B. The `ssh-keygen` utility generates SSH keys, including private keys for individuals, that can be used instead of passwords for remote logins to SSH servers. (These keys must be transferred and added to appropriate files on the server; using `ssh-keygen` alone isn't enough to accomplish the goal.) Option A's `authorized_keys` is a file in which keys are stored on a server; it's not a program to generate them. Option C's `sshpasswd` is fictitious. Option D's `id_rsa` is the file in which private keys are stored on the client; it's not a program to generate them.
13. D. SSH private keys are extremely sensitive, and they should not normally be distributed to anybody, since possession of another system's private key will make it easier for a miscreant to pretend to be you. Thus, option D is correct. It's safe to distribute *public* keys widely, as in any of the other options, and in fact SSH does this automatically; but *private* keys should remain just that: *private*.
14. B. PostScript is the de facto printing standard for Unix and Linux programs. Linux programs generally *do not* send data directly to the printer port (option A); on a multitasking, multiuser system, this would produce chaos because of competing print jobs. Although a few programs include printer driver collections (option C), most forgo this in favor of generating PostScript. Printing utilities come standard with Linux; add-on commercial utilities aren't required (option D).
15. A. The smart filter makes a print queue "smart" in that it can accept different file types (plain text, PostScript, graphics, and so on) and print them all correctly. It does not detect confidential information or prank print jobs, as suggested in options B and D. The `lpr` program in the CUPS printing system can be given a parameter to e-mail a user when the job finishes, but the smart filter doesn't do this, making option C incorrect.
16. B, D. The job ID and job owner are both displayed by `lpq`. Unless the application embeds its own name in the filename, that information won't be present (option A). Most printers lack Linux utilities to query ink or toner status; certainly `lpq` can't do this (option C).

17. D. The Foomatic package is a set of printer definitions. If it's not already installed, installing it may provide a driver for your printer. (Gutenprint is another package that may be helpful.) Options A and B are both alternatives to the CUPS Web-based configuration tool, but neither by itself provides access to additional printer definitions. The LPRng package of option C is an alternative to CUPS, but it doesn't provide extra printer drivers, so it's unlikely to help. Since LPRng uses older printing protocols, using it instead of CUPS could complicate your use of network-accessible printers.
18. C. The `smb://` part of the URI identifies a printer that uses the SMB/CIFS network protocol—the protocol used by Windows and Samba for file and printer sharing. Thus, option C is correct. The remaining fields were set to deliberately confuse; in this case, the computer is called `USB`, the printer uses the queue called `PARALLEL`, the username to access the server is `bsd`, and the password to access the printer is `lpd`. Most of these are likely to be poor choices in the real world, of course; they serve only to make the incorrect options A, B, and D seem more plausible—but these options remain incorrect.
19. A. The `-d` option to `lp` is the equivalent of `lpr's -P`; it selects the printer queue (`brother` in this case). Both utilities take the name of a file to print, so passing `sample.ps` works the same for both. The `lp` utility has no lowercase `-p` or `--printer` option, and the uppercase `-P` option sets a page range.
20. B. CUPS provides an option to enable users who don't own a print job to delete it, but this option is disabled by default. If the system administrator enables this option, users may cancel each others' print jobs, as option B states. Queue errors and original file permissions have no effect on this ability, contrary to options A and C. Option D directly contradicts the correct answer, option B.

Chapter 10

Configuring Network Servers I

THE FOLLOWING COMPTIA OBJECTIVES ARE COVERED IN THIS CHAPTER:

- ✓ **3.2 Implement interoperability with Windows using the following (Security and authentication [Kerberos]).**
- ✓ **3.8 Explain the purpose of each of the following mail services, protocols, and features (Protocols: SMTP, IMAP, POP3; MTA: Postfix, Sendmail; Email aliases: /etc/aliases, newaliases).**
- ✓ **3.10 Set up, install, configure, and maintain a BIND DNS server and related services (DNS utilities: named, rndc; Config file locations [/var/named]; Forward zones, reverse zones, root hints).**
- ✓ **3.11 Perform basic administration of the DHCP server (/etc/dhcpd.conf, dhcpd.leases).**
- ✓ **3.12 Given a scenario, troubleshoot NTP related issues (/etc/ntp.conf, ntpdate, date, ntpq -p).**
- ✓ **4.4 Explain the different DNS record types and the process of DNS resolution (Local resolution, TTL/caching, Root name servers, A, MX, PTR, CNAME, NS, TXT).**



Many Linux computers exist solely to function as server computers—they respond to access requests from other computers, delivering IP addresses, time data, e-mail, and other information to their clients. Many of the protocols and software packages that enable Linux to function as a server system are extremely complex, and entire books have been written about many of these protocols. Nonetheless, it's possible to get basic systems up and running without too much in-depth knowledge. This chapter and the next cover this topic for many common servers, so you should be able to get simple server configurations running.



Objective 3.2 is covered partly in this chapter and partly in Chapter 11, “Configuring Network Servers II.”

Delivering Network Information

Increasingly, local networks rely on several common protocols to help tie the network into a coherent whole. These protocols may be used across platforms—that is, Windows, Mac OS, Linux, and other clients may all access a Linux server. Examples of these protocols are the Dynamic Host Configuration Protocol (DHCP), the Domain Name System (DNS), and the Network Time Protocol (NTP). Linux can function as a server for each of these protocols. On many networks, one Linux system can run all of these protocols, although you can also split the duty up across multiple server computers, if you prefer.



Some of these protocols (particularly DNS and NTP) are used on the Internet at large, as well as on local networks. This chapter emphasizes their configuration as local network servers, since this type of configuration is simpler, more common, and less risky than is a public (Internet-accessible) configuration. You may want to employ a firewall, as described in Chapter 9, “Configuring Advanced Networking,” to block outside access attempts to all of these server ports.

Delivering IP Addresses with DHCP

Chapter 8, “Configuring Basic Networking,” described how to configure a computer to use an existing DHCP server to obtain its IP address. That DHCP server can run on any

number of OSs, including Linux. If you want Linux to do this, you must first install the DHCP server package, which is usually called `dhcp-server` or `dhcp`. This package normally includes a SysV startup script, such as `/etc/init.d/dhcpd`, which launches the server at system startup time. The server program itself is called `dhcpd` and is normally located in `/usr/sbin`.



The DHCP server program (`dhcpd`) has a very similar name to one of the three common Linux DHCP clients (`dhcpcd`). This similarity can be confusing; it's easy to install the wrong package or waste time trying to get the wrong daemon running.

The main DHCP server configuration file is `/etc/dhcpd.conf`. Listing 10.1 shows a sample of this file, which can serve as a starting point for a basic configuration. The file consists of two main parts: a series of global options and a subnet declaration that sets options for a particular subnet the DHCP server handles. (In a complex network, a single DHCP server might have multiple network interfaces, each with its own subnet declaration in `dhcpd.conf`.)

Listing 10.1: Sample DHCP Server Configuration

```
default-lease-time 86400;
max-lease-time 172800;
option subnet-mask 255.255.255.0;
option domain-name-servers 192.168.1.3, 10.128.60.8;
option domain-name "example.com";
option netbios-name-servers 192.168.1.3;
option netbios-node-type 8;
get-lease-hostnames true;
use-host-decl-names true;
ddns-update-style none;

subnet 192.168.1.0 netmask 255.255.255.0 {
    range 192.168.1.50 192.168.1.175;
    option routers 192.168.1.1;
}
```

Most of the options in Listing 10.1 set features that are self-explanatory or that you shouldn't need to change. Features you're most likely to want to adjust include the following:

Lease times The `default-lease-time` and `max-lease-time` options set the default and maximum DHCP lease times in seconds. For testing purposes, lease times of just a few minutes (perhaps 100–500 seconds) are reasonable. For a working network, lease times of several hours to several days (roughly 5,000–1,000,000 seconds) are more reasonable.



If you expect to make major changes to your network configuration that affect your DHCP server configuration, try reducing your lease times in stages prior to implementing these changes—for instance, reduce lease times from one week to one day to one hour, and perhaps to even lower values. The shorter lease times will ensure that your clients will check back more frequently as the network change time approaches, thus reducing problems caused by clients operating with outdated information. When the network changes are complete, increase the lease times to their original values to reduce the load on the DHCP server.

Network mask The option `subnet-mask` line sets the network mask delivered to clients.

Name servers You can point clients to one or more DNS name servers with the option `domain-name-servers` line. As in Listing 10.1, multiple servers should be separated by commas.

Domain name You can set the domain name with the option `domain-name` line. Clients might or might not use this information.

NetBIOS options You can provide several NetBIOS options to Windows systems with various options that include the string `netbios` in their name. Listing 10.1 points Windows systems to a NetBIOS name server and sets the NetBIOS node type. The values in Listing 10.1 are reasonable for most networks, although you must change the IP address of the NetBIOS name server for your network. If you don't know what your NetBIOS name server's IP address is, you should omit these lines.

Subnet declaration The `subnet` line begins the definition of a subnet the server is to handle. This begins with a specification of the subnet range, in the form of a network address (192.168.1.0), the `netmask` keyword, and a network mask (255.255.255.0). The network mask often matches the one specified with the option `subnet-mask` line, but it doesn't have to. An open curly brace (`{`) then begins the subnet declaration itself, in which you set options that apply only to this subnet. The subnet declaration ends with a close curly brace (`}`).

IP address assignments You tell the DHCP server what IP addresses to manage on the `range` line, which includes two IP addresses. The server assigns addresses within that range to any client that asks for one. Note that this range should *not* include the IP address used by the DHCP server itself or any other computer to which you assign a static IP address. (You can use DHCP to assign fixed IP addresses, even to servers, using more advanced configurations.) Be sure to include enough IP addresses in this range. Remember that you may need more addresses than you have computers, particularly if computers come and go (as in notebooks) or if virtual machines consume extra IP addresses.

Routers The option `routers` line sets the IP address of the router for this subnet.

Most lines in `dhcpd.conf` end in semicolons (`;`). The exceptions are lines that denote the start or end of a block of lines, such as the first and last lines in a `subnet` declaration.

To keep track of its leases, the DHCP server maintains a file, `dhcp.leases`, which holds lease information. This file is likely to reside in `/var/lib/dhcp` or `/var/db`. You shouldn't

need to edit this file, but you may want to consult it if you're having DHCP problems on your network.

The Linux DHCP server is very flexible and supports many options not described here, such as the ability to assign fixed IP addresses to specific computers. This can be handy if you want to configure server computers via DHCP. For more information, consult the `dhcpd.conf` man page or a book on DHCP, such as Ralph Droms and Ted Lemon's *The DHCP Handbook, 2nd Edition* (Sams, 2002).

Delivering Hostnames with DNS

Just as Linux uses a DNS server to resolve hostnames to IP addresses (and vice versa), Linux can run a DNS server program for the benefit of other systems. The most common Linux DNS package is the Berkeley Internet Name Domain (BIND), which installs a server under the file-name named. This server is typically started through a SysV startup script of the same name—like a DHCP server, a DNS server works best when it can run continuously to cache the names it serves, so DNS servers aren't typically run from super servers. There are two main parts to BIND configuration: setting global BIND options and configuring one or more domains that the server manages itself.



Real World Scenario

Obtaining a Domain

You may run a DNS server on a small local network without registering a domain name. Such a configuration normally uses a made-up domain name, ideally using a top-level domain (TLD, such as `.com` or `.edu`) that's not in use (such as `.invalid`); or you can configure the system as a forwarding-only server that doesn't handle any local hostnames.

You may also run a DNS server using a registered domain name. The Internet Corporation for Assigned Names and Numbers (ICANN) maintains a list of accredited registrars (<http://www.icann.org/en/registrars/accredited-list.html>). Check this list to locate a registrar for the TLD you want to use. You may then register a domain name, which normally costs about \$10 or \$15 a year, and set up a DNS server to manage that domain. Either you can link your own DNS server to the global DNS network by entering your server's IP address in Web forms maintained by your domain registrar, or you can have your registrar maintain your public (Internet-accessible) DNS presence and set up your own DNS server for your private network's use only. Which approach is best depends on your needs and resources. If you run a lot of servers or want to provide your users with direct access to your network's systems from off-site, running your own publicly accessible DNS server makes sense, although doing so increases your security risks. Running a private DNS server is sensible if your network is protected behind a network address translation (NAT) router or if there's no need to provide direct outside access to most of your network's computers. (Note that your computers may still be accessible via IP address even if no hostnames point directly to them, so failing to provide DNS entries is not an effective security measure.)

Setting Overall BIND Options

The main BIND configuration file is `/etc/named.conf`. This file controls overall server operation, including global options (in a section that begins with the keyword `options`) and one or more zone sections that point to files that describe the domains the server manages. The simplest type of BIND configuration is as a *forwarding-only DNS server*—that is, as a server that merely forwards DNS lookups to another computer. This configuration can improve DNS lookup times because your local DNS server can cache the names most frequently looked up by users on your local network, eliminating the need for your systems to consult an outside DNS server. A forwarding-only DNS configuration is enabled in the `options` section of `named.conf`, as shown in Listing 10.2. Most lines in this file end in a semicolon (`;`). This punctuation denotes the end of a configuration option. Comments are indicated by two leading slashes (`//`).

Listing 10.2: Sample BIND `/etc/named.conf` Configuration

```
options {
    directory "/var/named";
    forwarders {
        10.9.16.30;
        10.13.16.30;
    };
    listen-on{
        192.168.1.1;
        172.24.21.1;
    };
    forward only;
};
```

The key sections of this definition are the `forwarders` lines and the `forward only` line. The `forwarders` definition specifies the DNS servers to which BIND should forward the lookup requests it receives. This example specifies two outside systems; BIND will try each of them in turn, stopping when it receives a reply. The `forward only` line tells BIND that it should function *only* as a forwarding server; that is, it won't attempt to do lookups itself. If you change this line to `forward first`, then BIND will attempt to get an answer from the systems specified in the `forwarders` area, but if they don't answer, BIND will attempt to perform a *full recursive DNS lookup*. In this process, BIND contacts the *root DNS servers* (aka the *root name servers*), which know which systems handle the TLDs. These systems know which servers handle individual domain names, and so on. BIND queries each of these systems in turn. Full recursive lookups can take some time by computer standards (perhaps a couple of seconds), and the faster the network connection, the faster this happens. Thus, leaving this task to a server that's closer to the Internet at large than your own system is generally a good idea.

Listing 10.2 has a `listen-on` section, which tells BIND on which IP addresses it should listen. This option is most useful on server computers that have multiple network interfaces; you can have BIND respond to queries from some interfaces but not others.

Configuring the Root Zone

Performing a full recursive lookup requires that your system have an accurate root zone record. This is normally installed along with BIND and is referenced by the zone "." section of `named.conf` (which does not appear in Listing 10.2):

```
zone "." IN {
    type hint;
    file "named.ca";
};
```

These lines define a DNS zone—that is, a set of addresses that the server can resolve using rules defined elsewhere. The "." zone is also known as the *root zone*—this is the entire Internet namespace. This zone is defined by the `named.ca` file, which is located in the directory defined by the `directory` line in the `options` section of `named.conf`—normally `/var/named`. You shouldn't ordinarily change this particular definition, since it's normally the same for all DNS servers that use the Internet's DNS. If your system is configured to forward DNS requests, as described earlier in "Setting Overall BIND Options," the root zone definition is optional.

If the root servers ever change (as they do from time to time), your existing configuration might not work. You can retrieve the current root zone file in various ways. One is to type the following command:

```
$ dig @a.root-servers.net . ns > db.cache
```

Alternatively, you can download `db.cache` from `ftp://ftp.internic.net/domain/`. (Note that this is an FTP site, not a Web site. Chapter 11 provides some pointers on using FTP clients.) Once you've retrieved the latest zone file, you can replace the old one (`named.ca` in the preceding example; the filename you retrieve may not match what's used on your system) with the new file.

Configuring a Domain with BIND

On a small local network (say, one with a dozen or so computers), you can set up each computer with an `/etc/hosts` file for hostname resolution, as described in Chapter 8. This approach works, but it becomes tedious as the number of computers grows; every time you add a computer or change its IP address, you must change the `/etc/hosts` file for *every* computer on the network. Rather than deal with this hassle, you may want to configure a DNS server to handle local hostnames. Doing so requires making three sets of changes: adding a zone reference to `/etc/named.conf`, creating a forward zone file, and creating a reverse zone file.

Adding a Zone Reference to the Main BIND Configuration File

A default `/etc/named.conf` file usually includes one or more zone file references, such as the one to the root zone presented earlier. These references begin with the keyword `zone`, include the name of the domain to be managed, and link to a zone file in another directory (normally `/var/named`).

To have your system deliver IP addresses for local computers, you must create both forward and reverse zone files (as described shortly) and point your main `named.conf` file to your new definitions. Modifying `named.conf` is fairly straightforward; your changes will look something like this:

```
zone "pangaea.edu" {
    type master;
    file "named.pangaea.edu";
};
zone "1.168.192.in-addr.arpa" {
    type master;
    file "named.192.168.1";
};
```

The first four lines point BIND to the *forward zone* file, which enables the server to return IP addresses when it's fed hostnames. The final four lines point BIND to the *reverse zone* file, which enables the server to return hostnames when it's fed IP addresses. Both definitions should include the `type master` option and specify a file (normally in `/var/named`) in which the definition will be created. (You can also specify other types, as in `type slave`; however, these configurations require additional setup work not described here. In particular, slave configurations require at least two DNS servers, one of which copies the other's records.) The forward zone is named after the domain name it defines—`pangaea.edu` in this example. The reverse zone is named after the IP address block it serves but with a couple of twists. First, the `.in-addr.arpa` string is added to the end of the IP address range. Second, the IP address range's numbers are reversed. For instance, if you're defining a reverse zone file for `192.168.1.0/24`, you'd take the network portion of the address (`192.168.1`), reverse its numbers (`1.168.192`), and add `.in-addr.arpa`, resulting in `1.168.192.in-addr.arpa`.

Creating a Forward Zone File

A forward zone file creates the mapping of hostnames to IP addresses. Listing 10.3 shows an example. This file might be the `named.pangaea.edu` file referenced earlier.

Listing 10.3: Sample Forward Zone Configuration File

```
$TTL 1D
pangaea.edu.      IN  SOA  dns1.pangaea.edu. \
                    admin.pangaea.edu. (
                        2010022003 ; serial
                        3600        ; refresh
                        600         ; retry
                        604800      ; expire
                        86400       ; default_ttl
                    )
dns1.pangaea.edu. IN  A      192.168.1.1
```



```

coelophysis.pangaea.edu.  IN A      192.168.1.2
peteinosaurus            IN A      192.168.1.3
                        IN A      192.168.1.4
pangaea.edu.            IN A      192.168.1.5
www                     IN CNAME  webhosting.example.com.
ftp                     IN CNAME  coelophysis
@                       IN MX      10 peteinosaurus
@                       IN MX      20 mail.example.com.
@                       IN NS      dns1.pangaea.edu.

```

Most of the lines in a zone configuration file take the following form:

```
name IN record-type record-contents
```

name is the name of the computer (or a name derived from the computer's address, in the case of reverse zone configuration files, as described shortly). You should pay careful attention to the trailing dots (.) in the fully qualified domain names (FQDNs) in Listing 10.3—the names that include both machine name and domain name portions. Technically, all DNS hostnames end in dots, although they can usually be omitted when you're using Web browsers, e-mail clients, and most other tools. In the case of a zone file, though, the dots are mandatory when you specify a complete hostname. When a name does *not* end in a dot, BIND adds the current zone name—*pangaea.edu.*, in the case of Listing 10.3. Thus, you can specify hostnames either completely (as in *coelophysis.pangaea.edu.*) or by host portion only (as in *peteinosaurus*). If you omit a name completely (as in 192.168.1.4 in Listing 10.3), the previous hostname (*peteinosaurus* in Listing 10.3) is linked to both IP addresses (192.168.1.3 and 192.168.1.4 in Listing 10.3). The DNS server delivers all the IP addresses that are so linked in a round-robin fashion. This is an easy way to perform load balancing—if you need two computers to handle the load for one Web site or other server, you can configure two computers identically and have the DNS server direct half the traffic to each computer. The at-sign (@) is a stand-in for the domain itself; it's commonly used with NS and MX records, as shown in Listing 10.3.

Following the hostname comes the string IN, which stands for *Internet*. Next is a code for the record type. Several record types are common and important:

A An address (A) record links a hostname to an IPv4 address. You may specify the hostname as an FQDN or as a hostname without its domain portion, as described earlier. You may also assign an IP address to the domain name alone (as in *pangaea.edu.* in Listing 10.3).

AAAA These records are the IPv6 equivalent of A records.

CNAME A canonical name (CNAME) record links a hostname to another hostname. You may specify the *record-contents* either as a “bare” machine name or as an FQDN. In the latter case, the target system need not be in the same domain as the one specified by the SOA record. For instance, in Listing 10.3, *www* links to an outside system.

NS A name server (NS) record provides the hostname of a DNS server for the domain. This record is used mainly by other DNS servers that are directed to yours by other systems

when your DNS server functions on the Internet at large. NS records, like CNAME records, can point outside your own domain.

MX A mail exchanger (MX) record points to a mail server for the domain. Remote mail servers access this record to learn how to deliver mail that's addressed to your domain. The *record-contents* of these records includes both a priority code (10 or 20 in Listing 10.3; sending servers try to contact servers with lower priority codes first) and a computer name. As with CNAME and NS records, MX records can point to systems in other domains. The mail server must be configured to accept mail for the domain, though.

PTR Pointer (PTR) records are the opposite of A records; they link IP addresses to host-names. As such, they don't exist in Listing 10.3. The upcoming section "Creating a Reverse Zone File" covers this type of record in more detail.

SOA The start of authority (SOA) record is the first one in Listing 10.3. This type of record is complex enough that I describe it in more detail shortly.

TXT This record type enables you to set explanatory text associated with the domain.

Most BIND packages for Linux include one or more forward zone files, such as one for the `localhost` name and perhaps a rudimentary file (called `named.empty` or something similar) that you can use as a model for your own domain. You may also use Listing 10.3 as a model.

The SOA record is particularly complex. It provides various administrative details for the zone. The *name* is the domain name for the record, and this name is used as the default whenever a machine name without a trailing dot (.) appears in subsequent records. An at-sign (@) can function as a stand-in for the domain name; in that case, the zone name specified in the `/etc/named.conf` file is used instead.

The *record-contents* field of an SOA record is itself quite complex. It includes several components, separated by spaces:

- The name of the primary name server (which should also have its own NS record).
- The e-mail address of the domain's administrator, but with a dot (.) instead of an at-sign (@). For instance, Listing 10.3's domain administrator is `admin@pangaea.edu`.
- A set of numbers within commas specifying values that other DNS servers use to determine how long to cache records for the domain. (Cache times are sometimes referred to as a time-to-live, or TTL, value.) Times are specified in seconds, although you may append M, H, D, or W to specify times in minutes, hours, days, or weeks, respectively. Cache times for working domains are usually set in the range of hours to days; however, when testing a configuration, you might want to set cache times of just a minute or two. You might also want to lower the cache times on a working domain a few days prior to making major changes to your domain so that clients won't be delivering outdated information after you make your changes.

The serial number (the line flagged as `serial` in Listing 10.3) is particularly important on networks that employ both master and slave DNS servers. Slave servers examine this value in zone files to determine whether to update their local versions of the zone file. If the serial number doesn't change, the slave server won't update its local file. This fact can cause

confusion because if you forget to update your serial number, changes to your zone file may not propagate properly. You can use any system you like for creating serial numbers. Listing 10.3 uses a date with an appended revision number—2010022003 refers to the third change (03) on February 20, 2010. A simple incrementing number will work, too (1, then 2, then 3, and so on). Be sure your numbers increment in a strictly linear way—if you use the date, be sure to use *YYYYMMDD* format.

Creating a Reverse Zone File

Reverse zone files are conceptually similar to forward zone files; however, they use PTR records rather than A records, and they map IP addresses to hostnames. They do this by using hostnames in the *in-addr.arpa* domain (or *ip6.arpa* in the case of IPv6 addresses) as a way to represent IP addresses in hostname form. Reverse zone files also often lack some common features of forward zone files, such as MX listings. Listing 10.4 presents an example of a reverse DNS zone file. This listing might be used in conjunction with Listing 10.3.

Listing 10.4: Sample Reverse Zone Configuration File

```
$TTL 1D
1.168.192.in-addr.arpa.  IN SOA  dns1.pangaea.edu. \
                        admin.pangaea.edu. (
                        2010022003 ; serial
                        3600      ; refresh
                        600       ; retry
                        604800    ; expire
                        86400     ; default_ttl
                        )
1.1.168.192.in-addr.arpa. IN PTR   dns1.pangaea.edu.
2                        IN PTR   coelophysis.pangaea.edu.
3                        IN PTR   peteinosaurus.pangaea.edu.
4                        IN PTR   peteinosaurus.pangaea.edu.
5                        IN PTR   pangaea.edu.
@                        IN NS    dns1.pangaea.edu.
```

The hostnames in a reverse zone file (in the *name* fields of the file's entries) are based on the IP addresses, as described earlier in “Adding a Zone Reference to the Main BIND Configuration File.” Listing 10.4 provides listings for the 192.168.1.0/24 domain, so its SOA field specifies *1.168.192.in-addr.arpa.* as the domain name. As with forward entries, the *name* field may present either a complete hostname, including a trailing dot (*.*), or just the part of the hostname that comes to the left of the domain portion of the name. The latter is normally the final one to three numbers in the IP address, but in reverse order. The domain name in PTR records ends in either *in-addr.arpa.* (for IPv4 addresses) or *ip6.arpa.* (for IPv6 addresses).

The *record-contents* portion of a reverse zone file's PTR entries consists of DNS hostnames, complete with the domain portion and trailing dot (*.*), as illustrated in Listing 10.4. If

you omit the DNS domain name, the record will be misinterpreted as being part of the domain defined by the reverse zone file—`1.168.192.in-addr.arpa`, in the case of Listing 10.4.

Running a DNS Server

A single DNS server can manage multiple domains. To do so, you must create one zone file for each domain managed (forward zones and reverse zones each require their own files) and reference each of them in `/etc/named.conf`. If these zones are to be publicly accessible, you must also register them with the upstream DNS servers, via your domain registrar.

With the `/etc/named.conf` file and all the zone files configured, you can launch your DNS server or tell it to reread its configuration files. You normally do this via its SysV startup script, as described in Chapter 4, “Managing System Services.” This will launch the BIND server, which is normally called `named`. You may want to check your system log files (also described in Chapter 4) at this point; it’s not uncommon for a misconfiguration to cause the DNS server to fail to start or to misbehave in some other way. Such problems often leave traces in log files. If the server seems to be running, you can check its operation using `named`, `host`, or `dig`, as described in Chapter 8, “Configuring Basic Networking.” Be sure to test the system using the server computer itself, another computer on the local network that’s configured to use your DNS server directly, and (if applicable) a remote system. Remote computers, though, may not immediately use your DNS server, since it can take minutes, hours, or sometimes even days for DNS changes to propagate through the entire DNS network.

Once it’s running, you may control the `named` server using the `rndc` utility. You pass the utility any of several commands on the command line. For instance, typing **`rndc reload`** reloads the configuration files, **`rndc stop`** terminates the server, **`rndc flush`** flushes the server’s caches, and **`rndc status`** displays the server’s status information. Typing **`rndc`** alone presents a complete list of the commands it accepts.

Delivering the Time with NTP

Typically, a clock on an isolated computer needn’t be set with any great precision. It doesn’t really matter if the time is off by a few seconds, or even a few minutes, so long as the time is reasonably consistent on that one computer for the purpose of `cron`, other scheduling tools, and files’ time stamps. Sometimes, though, maintaining a truly accurate system time is important. This is true for a few scientific, business, and industrial applications (such as astronomical measurements or a system that determines the start and stop times for television broadcasts).

In a networked environment, maintaining the correct time can be more important. Time stamps on files may become confused if a file server and its clients have different times, for instance. Worse, a few protocols, such as the Kerberos security suite, embed time stamps in their packets and rely on those time stamps for normal system functioning. If two systems using Kerberos have wildly different times, they may not be able to communicate. For these reasons, several protocols exist to synchronize the clocks of multiple systems. Of these,

NTP is the most popular and flexible. You should first understand the basic principles of NTP operation. You can then go on to configuring an NTP server for your network and setting up other systems as NTP clients.



Other time-setting protocols include one built into the Server Message Block/Common Internet File System (SMB/CIFS) used for Windows file sharing and implemented in Linux by Samba and a protocol used by the `rdate` utility in Linux.

Understanding NTP Basics

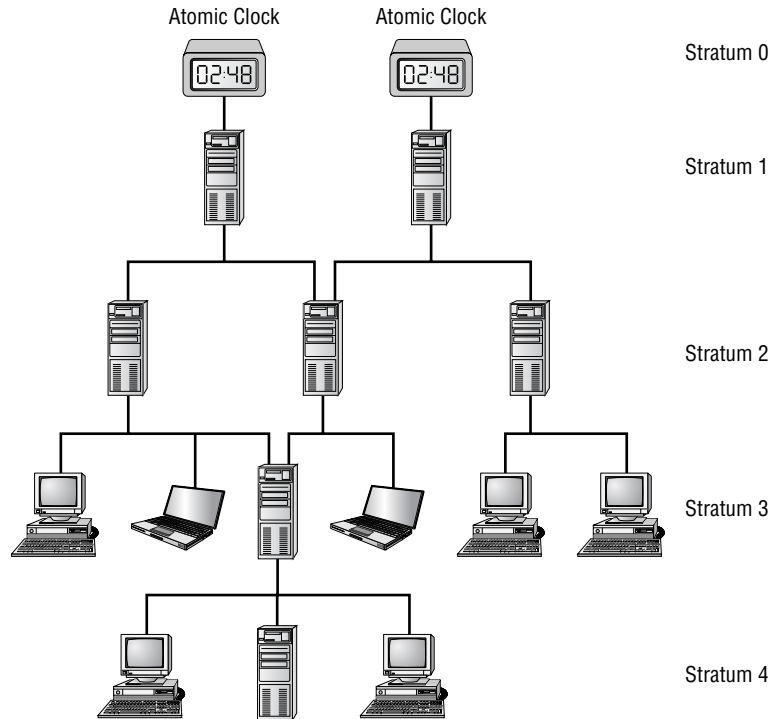
Linux uses the `date` command to enable you to view or set the date and time. Typing **date** alone displays the current date and time. You can set the date and time by adding them to the command line in `MMDDhhmm[[CC]YY][.ss]` form. The `date` command is, however, not network-enabled, so setting the time using this tool is labor-intensive. NTP solves that problem by enabling one computer to set its clock based on another one.

NTP creates a tiered hierarchy of time sources, as illustrated in Figure 10.1. At the top of the structure are one or more highly accurate time sources—typically atomic clocks or radio receivers that pull their times from broadcast time signals based on atomic clocks. These are referred to as *stratum 0* time servers, but they aren't directly accessible to any but the *stratum 1* time servers to which they're connected. These stratum 1 computers run NTP servers that deliver the time to *stratum 2* servers, which deliver the time to *stratum 3* servers, and so on, for an arbitrary number of strata.

The key to NTP is the fact that each server can deliver time to an expanding number of clients. For instance, if a stratum 1 server has 1,000 clients, each of which has 1,000 clients, and so on, then stratum 3 will consist of 1,000,000 systems, and stratum 4 will contain 1,000,000,000 systems. Each increase in the stratum number slightly decreases the accuracy of the time signal, but not by much; even a stratum 4 system's clock should be accurate to well under a second, which is accurate enough for almost all purposes. More important, if you run a network, you can set aside one computer as an NTP server and set all your other computers' clocks from that one server. Even if your primary NTP server's clock is off by a second, all the clocks on your network should be set to within a tiny fraction a second of each other, which is the most important consideration for time-dependent network protocols such as Kerberos.

NTP works by measuring the round-trip time for packets between the server and the client. The two systems exchange packets with embedded time stamps; the client then adjusts its time so that it is synchronized with the source's time stamp but adds a bit to the time reported by the source to account for this round-trip delay. For this reason, when you select an NTP source (as described next, in "Locating a Time Source"), you should pick one with the shortest possible network time delay, all other things being equal. (In truth, several measures of reliability exist, and the NTP programs try to take them all into account.)

FIGURE 10.1 NTP enables an expanding pyramid of computers to set their clocks to a highly accurate source signal.



The main Linux NTP server program functions as both a server and a client; it sets its clock based on the time of the server to which it's pointed, and it enables other systems to set their clocks based on its own. Even the end points in the NTP hierarchy (the stratum 4 and some stratum 3 servers in Figure 10.1) often run the full NTP server package. The reason is that this software runs constantly and can monitor for and adjust the clock drift that's common in computers' clocks, resulting in much more consistent timekeeping than is possible with a program that simply sets the clock and then ignores it until the next time the program is run. In other words, NTP doesn't just reset the system clock periodically; the server improves the accuracy of the system clock. In part, this is done through the `ntp.drift` file, which is usually buried in `/var/lib/ntp` but is sometimes stored in `/etc`. This file holds information about the software clock's inaccuracies and so can be used to correct for them. A full NTP server, even when it's functioning only as an NTP client, periodically checks with its source servers to keep the system time set correctly and to update the `ntp.drift` file.

Locating a Time Source

You may think that locating an NTP server with a low stratum number (such as stratum 1) is ideal. Although it's true that your own system will have a minutely more accurate clock when using such a source, the best approach in most cases is to synchronize with a stratum 2 or lower system. The reason is that this practice will help keep the load on the stratum 1 servers low, thus improving the overall performance of the NTP network as a whole. An exception might be if you're configuring an NTP server that will itself deliver the time to hundreds or more computers.

To locate an NTP server, you should consult one or more of several sources:

Your ISP Many Internet service providers (ISPs), including business networks and universities, operate NTP servers for the benefit of their users. These servers are usually very close to your own in a network sense, making them good choices for NTP. You should consult your ISP or the networking department at your organization to learn whether such a system is available.

Your distribution's NTP server Some Linux distributions operate NTP servers for their users. If you happen to be close to these servers in a network sense, they can be good choices; however, chances are this isn't the case, so you may want to look elsewhere.

Public NTP server lists Lists of public NTP servers are maintained at <http://support.ntp.org/bin/view/Servers/WebHome>. These servers can be good choices, but you'll need to locate the one closest to you in a network sense and perhaps contact the site you choose to obtain permission to use it.

Public NTP server pool The `pool.ntp.org` subdomain is dedicated to servers that have volunteered to function as public NTP servers. These servers are accessed in a round-robin fashion by hostname, so you can end up using different servers each time you launch NTP. Thus, using the public NTP server pool can be a bit of a gamble, but the results are usually good enough for casual users or if you don't want to spend time checking and maintaining your NTP configuration. To use the pool, you can configure your NTP server to use either the `pool.ntp.org` subdomain name or a numbered host within that domain, such as `0.pool.ntp.org`. Consult <http://support.ntp.org/bin/view/Servers/NTPPoolServers> for details.



The closest server in a network sense may not be the closest computer in a geographic sense. For instance, a national ISP may route all traffic through just one or two hub sites. The result can be that traffic from, say, Atlanta, Georgia, to Tampa, Florida, may go through Chicago, Illinois. Such a detour is likely to increase round-trip time and decrease the accuracy of NTP. In such a situation, a user in Atlanta may be better off using a Chicago NTP server than one in Tampa, even though Tampa is much closer geographically.

Once you've located a few possible time servers, try using `ping` to determine the round-trip time for packets to this system. If any systems have very high `ping` times, you may want to remove them from consideration.



Real World Scenario

Serving Time to Windows Systems

If your network hosts both Linux and Windows systems, you may want to use a Linux system as a time source for Windows clients or conceivably even use a Windows server as a time source for Linux clients. One way to do this is to run NTP on Windows. Consult <http://www.meinberg.de/english/sw/ntp.htm> or perform a Web search to locate NTP software for Windows systems. For Windows NT/200x/XP/Vista, you can type **NET TIME /SETSNTP:time.server**, where *time.server* is the name of your local NTP time server. This command performs a one-time setting of the clock but doesn't run in the background as the full NTP package does on Linux. Running this command in a Windows login script may be adequate for your purposes.

For older Windows 9x/Me systems, you can type **NET TIME \\SERVER /SET /YES** to have the system set the time to the time maintained by *SERVER*, which must be a Windows or Samba file or print server. This command doesn't use NTP, but if you have a Linux system that runs both NTP and Samba, it can be a good way to get the job done.

Configuring NTP Servers

When you're setting up a network to use NTP, select one system (or perhaps two for a network with several dozen or more computers) to function as the primary NTP server. This computer needn't be very powerful, but it must have always-up access to the network. You can then install the NTP server and configure it.

Most Linux distributions ship the NTP software in a package called `ntp`, `xntp`, `ntpd`, or `xntpd`. Look for this package and, if it's not already installed, install it. If you can't find this package, check <http://www.ntp.org/downloads.html>. This site hosts NTP source code, which you can compile and install. Alternatively, you can look for a binary package for another distribution. In any event, if you don't install your distribution's own NTP package, you'll need to create your own SysV startup script or start the NTP daemon in some other way.

Once NTP is installed, look for its configuration file, `/etc/ntp.conf`. This file contains various NTP options, but the most important are these server lines:

```
server clock.example.com
server ntp.pangaea.edu
server time.luna.edu
```

Each of these lines points to a single NTP server. When your local NTP daemon starts up, it contacts all the servers specified in `/etc/ntp.conf`, measures their accuracy against each other, and settles on one as its primary time source. Typically, you list about three upstream time servers for a system that's to serve many other computers. This practice

enables your server to weed out any servers that deliver a bad time signal, and it also gives automatic fallback in case an upstream server becomes temporarily or permanently unavailable. If your NTP server won't be serving many computers itself, you may want to configure it for three servers initially and then drop the ones your system isn't using as its primary time source after a day or two. This will reduce the load on these servers.

You may want to peruse your configuration file for entries to remove. For instance, the configuration file may contain references to servers you'd rather not use or other odd options with associated comments that make you think they're inappropriate. Generally speaking, you shouldn't adjust entries in the `ntp.conf` file other than the reference server lines, but special circumstances or odd starting files may require you to make changes.

Once you've made your changes, start or restart your NTP daemon. Typically, this is done via a SysV startup script:

```
# /etc/init.d/ntp restart
```

You may need to change the path to the file, the SysV script filename, or the option (change `restart` to `start` if you're starting NTP for the first time). Most distributions configure NTP to start whenever the system boots once you install the server. Consult Chapter 4 for details on changing this configuration.

To verify that NTP is working, you can use `ntpq`, which is an interactive program that accepts various commands. Figure 10.2 shows it in operation, displaying the output of the `peers` command, which summarizes the servers to which your NTP server is connected. (Typing `ntpq -p` causes the program to display this list and then exit.) In Figure 10.2, three external servers are listed, plus `LOCAL(0)`, which is the last-resort reference source of the computer's own clock. The `refid` column shows the server to which each system is synchronized, the `st` column shows the stratum of the server, and additional columns show more technical information. The server to which yours is synchronized is denoted by an asterisk (*), other servers with good times are indicated by plus signs (+), and most other symbols (such as x and -) denote servers that have been discarded from consideration for various reasons. Consult `ntpq`'s man page for more information about its operation.

FIGURE 10.2 The `ntpq` program enables you to verify that an NTP server is functioning correctly.

```
rodsmith@speaker:~ - Shell - Konsole
Session Edit View Bookmarks Settings Help

$ ntpq
ntpq> peers
ntpq> remote      refid      st t when poll reach  delay  offset  jitter
=====
+clock.example.c 128.227.205.3 2 u 186 1024 377 23.016 -0.624 10.787
*ntp.pangaea.edu 172.17.1.243 2 u 269 1024 377 41.464 4.139 2.553
+time.luna.edu 18.145.0.30 2 u 186 1024 377 20.183 -0.293 0.366
LOCAL(0) LOCAL(0) 10 1 23 64 377 0.000 0.000 0.001
ntpq>
```

**NOTE**

You won't see a server selected as the source until a few minutes after you restart the NTP daemon. The reason is that your local NTP process takes a while to determine which of the sources is providing the best signal.

Configuring NTP Clients

Once you've configured one or more NTP servers, you can configure the rest of your systems to point to them. Their configuration is done just like the NTP server configuration, with a couple of exceptions:

- You set your NTP clients to refer to the NTP server (or servers) you've just configured rather than to an outside NTP source. This way, your local systems won't put an additional burden on the outside NTP server you've selected.
- You may want to ensure that your NTP clients can't be accessed as servers. This is a security measure. You can do this with an `iptables` firewall rule or by using the `restrict default ignore` line in `ntp.conf`. This line tells the server to ignore all incoming NTP requests. Ideally, you should use both methods. (If your distribution employs such tools by default, you may have to undo them on any system that should serve time to its own clients.)

Once you've configured a client, restart its NTP daemon. You can then use `ntpq` to check its status. You should see that it refers only to your network's own NTP server or servers. These systems should be listed as belonging to a stratum with a number one higher than the servers to which they refer.

In some cases, a simpler way to set the time on a client is to use `ntpdate`. This program is part of the NTP suite, and it performs a one-time clock setting. To use it, type the command name followed by the hostname or IP address of an NTP server:

```
# ntpdate clock.example.com
```

Some NTP packages include a call to `ntpdate` in their NTP daemon startup scripts in order to ensure that the system is set to the correct time when it starts. The `ntpdate` command, however, has been deprecated and could disappear from the NTP package at any time. Instead, you can start `ntpd` with its `-g` and `-q` options, which enable it to perform a one-time clock setting to a value that's wildly divergent from the current time. (Ordinarily, `ntpd` exits without adjusting the clock if the time server's time differs from the local time by more than a few minutes.)

Authenticating Users on the Network

One of the problems with large computer networks is that user authentication can become a problem. If a network has 100 computers, and if even just a few users need to be able to use any of these computers, managing the accounts for those users can become a full-time job

all by itself. The solution to this problem lies in network authentication protocols, including the following:

Kerberos This protocol, named after the mythological three-headed dog that guarded the underworld, provides encryption and authentication for network servers that explicitly support it. Typically, a Kerberos network uses a Key Distribution Center (KDC; a Kerberos server) along with Kerberos-enabled (aka Kerberized) servers and clients for specific protocols. Kerberos typically manages specific protocols rather than replace the standard Linux account database, although it can be tied into the standard login tools.

LDAP The Lightweight Directory Access Protocol (LDAP) is a tool for storing *directories*, which in this context are similar to databases. Among other things, these directories can include Linux user account data. LDAP supports encrypted or unencrypted authentication, using a central server and LDAP support in the Linux Pluggable Authentication Module (PAM) system. Used in this way, an LDAP server can supplement or replace Linux's normal account database. LDAP is a very flexible tool, capable of far more than simple network authentication.

NIS Network Information Services (NIS) is a tool that's similar in broad strokes to LDAP, although NIS predates LDAP and is geared toward providing Unix/Linux data, including account and network data.

Winbind The Linux Winbind server (aka `winbindd`, the name of the server program) is similar to LDAP and NIS in that it can supplement or replace the standard Linux account database. Winbind differs in that it uses a Windows domain controller (either an NT domain controller or a newer Active Directory [AD] controller) as its source of account information. This configuration imposes certain compromises; for instance, Windows domain controllers don't hold Linux username-to-UID mappings, so Winbind must manage this detail itself. Nonetheless, Winbind is very useful on Windows-dominated networks.

Chapter 12, "Securing Linux," provides additional details concerning NIS and LDAP. Fully configuring any of these tools is an advanced task, though, so you should consult additional documentation if you need to use network authentication.

Using E-mail

E-mail enables users to send messages to one another, across an office or across the world. Before configuring mail programs, you should understand the protocols involved. With that knowledge in hand, you can perform some basic mail server configuration. (I cover only the bare-bones basics for a few popular servers in this chapter.) You should also know how to set up e-mail forwarding and aliases, which are tools to redirect the delivery of e-mail from one account or address to another.

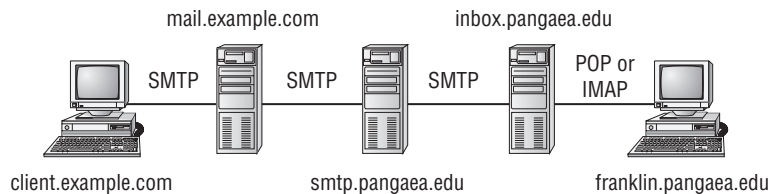
Understanding E-mail Protocols

Internet mail delivery today is dominated by a protocol known as the *Simple Mail Transfer Protocol (SMTP)*. This protocol is an example of a *push protocol*, meaning that the sending system initiates the transfer.

A user writes a message using a mail reader and then tells the mail reader to send the mail. The mail reader contacts a local SMTP server, which may run on the same or another computer. The SMTP server accepts the message for delivery, looks up the recipient address, and delivers the message to the recipient system.

In some cases, the recipient system may forward the mail to another system, which handles the mail account for the addressee. Depending on how the recipient reads mail, that person may use the destination mail server computer directly or run a mail client on another computer. In the latter case, the mail client uses a protocol such as the *Post Office Protocol (POP)* or the *Internet Message Access Protocol (IMAP)* to retrieve the mail from the local mail server. POP and IMAP are both examples of *pull protocols*, in which the recipient, rather than the sender, initiates the data transfer. This configuration is outlined in Figure 10.3. The Internet's mail system is flexible enough that the total number of links between the sender and recipient may be more or less than the number depicted in Figure 10.3, though. Users may read mail on the final SMTP server (inbox.pangaea.edu in Figure 10.3) or use a POP or IMAP client (as depicted in Figure 10.3) to read mail from another system. It's even possible to use a program such as Fetchmail (<http://fetchmail.berlios.de>) to extend the chain further after a POP or IMAP retrieval.

FIGURE 10.3 E-mail typically traverses several links between sender and recipient.



Three of the computers in Figure 10.3—mail.example.com, smtp.pangaea.edu, and inbox.pangaea.edu—must run SMTP servers. These servers can be entirely different products running on different platforms. For instance, one might be sendmail on Linux, another might be Postfix on FreeBSD, and a third might be Microsoft Exchange on Windows. In addition to running an SMTP server, Figure 10.3's inbox.pangaea.edu must run a POP or IMAP server. The two end-point computers—client.example.com and franklin.pangaea.edu—need not run mail servers. Instead, client.example.com connects to the SMTP server on mail.example.com to send mail, and franklin.pangaea.edu connects to the POP or IMAP server on inbox.pangaea.edu to retrieve mail.



Mail servers are sometimes called *mail transfer agents (MTAs)*, and mail clients (user systems) are sometimes called *mail user agents (MUAs)*.

Configuring SMTP Servers

Even Linux computers that don't exist as mail server computers often run mail server software. The reason is that certain local tools sometimes assume that a local mail server program will be present; these tools use the local mail server to deliver notices about their activities to root or to some other user. For instance, cron (described in Chapter 3) e-mails the output of the programs it runs to the user who runs them. Thus, basic e-mail configuration is often important, even on Linux systems that aren't primarily mail servers. The next few pages describe two common Linux SMTP server programs, sendmail and Postfix, as well as some of the commands and tools you can use to manage a mail queue on Linux.



SMTP servers can be misconfigured to function as *open mail relays*. These will forward mail from any address to any other address, and they are beloved by those who send *spam*—unsolicited bulk e-mail. All Linux distributions released since 1999 or so are configured to *not* be open mail relays by default. If you're running an older distribution or if you attempt to change your mail server's configuration, you should ensure that you aren't running an open mail relay. Consult http://www.mail-abuse.com/an_sec3rdparty.html for more information on this important topic.



This section can only scratch the surface of e-mail configuration, particularly for large mail server computers. For more information on mail server configuration, consult the server's own documentation or a book on the subject, such as Craig Hunt's *Linux Sendmail Administration* (Sybex, 2001) or Kyle D. Dent's *Postfix: The Definitive Guide* (O'Reilly, 2003).

Configuring Sendmail

The sendmail program (<http://www.sendmail.org>) has long been the most common mail server program on the Internet, although it has declined in popularity in recent years. Several Linux distributions use sendmail as the default mail server program. Most Linux distributions that use it provide it in a package called `sendmail`, so you can check to see whether that package is installed on your system.



The presence of a binary program called `sendmail` might not indicate the presence of the sendmail server. Many programs assume that the mail server executable is called `sendmail`, so other mail server packages usually provide a binary or link of that name for compatibility purposes.

The main sendmail configuration file is `sendmail.cf`, which is usually kept in `/etc/mail`. This file has a very complex and confusing structure, though. In practice, most

administrators write their sendmail configurations in another file, which is converted to a `sendmail.cf` file via a special utility, called `m4`:

```
# m4 < myconfig.mc > sendmail.cf
```



If you issue the `m4` command in the same directory in which the original `sendmail.cf` file resides, the command copies over the existing `/etc/mail/sendmail.cf` file. For added safety, back up that file first. You can then restore it from the backup if something goes wrong.

This command converts the `myconfig.mc` file into the `sendmail.cf` file. Where do you start, though? That is, where can you find a file to modify into `myconfig.mc`? Distributions that use sendmail typically provide sample configurations called `sendmail.mc`, `linux.smtp.mc`, or something similar. These files may exist in the `/etc/mail` directory or elsewhere. This file may be installed as part of the main sendmail package or as part of a separate package, such as Fedora's `sendmail-cf` package. You may also need to install the `m4` package, which holds the `m4` utility used to convert the `.mc` file to a `.cf` file.

For the most part, a typical desktop Linux system needs few or no changes to its sendmail configuration; the default values should work acceptably. Most distributions ship with sendmail configurations that cause the server to accept mail only from the computer on which the server runs. If you want sendmail to accept mail from other computers, though, you'll need to modify the configuration. To do so, look for a line like this:

```
DAEMON_OPTIONS(`Port=smtp,Addr=127.0.0.1, Name=MTA')dn1
```



The character before `Port` in this line isn't an ordinary single quote mark; it's an *open* single-quote mark, which can be typed from the key to the left of the number `1` key on most keyboards that also includes the "tilde." This character is also referred to as a "backtick" or a "back quote." In some fonts, the result looks like "curly" single quotes around the options within the parentheses.

This line tells sendmail to bind only to the `127.0.0.1` address—that is, the localhost interface. To have sendmail accept mail from other systems, you must comment this line out. The `.mc` file uses the string `dn1` as a comment indicator, so you should add that string to the start of the line:

```
dn1 DAEMON_OPTIONS(`Port=smtp,Addr=127.0.0.1, Name=MTA')dn1
```

You can then create a new `sendmail.cf` file by using `m4`, as just described. After you restart sendmail by using its SysV startup script, the server should accept mail from other computers.

Configuring Postfix

Postfix (<http://www.postfix.org>) isn't as popular on the Internet at large as sendmail, but it's now the default mail server for several Linux distributions, such as Mandriva and SUSE. On the whole, Postfix is simpler to configure than is sendmail; Postfix uses a single configuration file, `/etc/postfix/main.cf`, for most options, and the Postfix options are named more intuitively than are most sendmail equivalents. The default `main.cf` file is also copiously commented, so you can learn a great deal about Postfix configuration by reading that file.

As with sendmail, a default Postfix configuration works reasonably well for a stand-alone workstation or a nonmail server system. Most default Postfix configurations accept mail directed at the server computer from other systems, so reconfiguring it as described for sendmail isn't likely to be necessary. If you can't seem to send to the Postfix server from another computer, though, or if you want to close it off so that it rejects such access attempts, look for the `inet_interfaces` option:

```
inet_interfaces = $myhostname, localhost
```

This setting tells Postfix to listen on the network interface associated with `$myhostname` (which is set earlier to the computer's hostname) and to the localhost interface. You can remove `$myhostname` to have Postfix listen only on the localhost interface, or you can add it if it's not present and you want the server to listen on that interface.

After you make changes to the Postfix configuration, you can tell the server to immediately implement the changes:

```
# postfix reload
```

This command begins an orderly rereading of configuration files, and the various processes associated with Postfix restart at their earliest convenience. Using the SysV startup script's `restart` or `reload` option should have a similar effect.

Managing Mail Queues

The `sendmail` program functions both as a daemon and as a command that can accept mail for delivery as well as manage mail queues. In fact, some Linux programs send mail by calling the `sendmail` program, which is why Postfix and other Linux mail servers typically provide a program of the same name and that accepts the same options as the original program.

One of the most basic ways to use the `sendmail` command is to use its `-bp` option, which lists the mail messages that are still waiting to be sent. An equivalent command is `mailq`. In either form, this command is useful if you're not sure whether the mail server is delivering mail. For instance, if your configuration is bad or if a network connection is down, typing **mailq** should reveal a backlog of old messages. Such a listing might even provide clues to the nature of a problem. For instance, if mail to some sites is being delivered but mail to other sites isn't getting out, it could be a problem with routers, overzealous anti-spam configurations on some remote sites, or something about your own configuration that's tripping anti-spam alarms on the remote site.



Seeing messages in a mail queue listing isn't necessarily a sign of trouble. If your system is processing very many or very large messages, they will appear in the queue for a time. Likewise, a slow network connection will cause messages to hang around for a while. If messages regularly stay in the queue for very long, though, your network connection may be unreliable or overloaded, or something about your mail server software's configuration may be suboptimal. Checking the mail log files (typically `/var/log/mail`) may provide you with more clues.

If mail has accumulated in the queue and you believe you've corrected the problem, it should eventually clear out on its own. To speed up the process, though, you can type **sendmail -q**. This command causes the mail server to immediately attempt delivery of all queued messages.

Using Aliases and Forwarding E-mail

E-mail *aliases* enable one address to stand in for another one. For instance, all e-mail servers are supposed to maintain an account called `postmaster`. E-mail to this account should be read by somebody who's responsible for maintaining the system. One way to do this is to set up an alias linking the `postmaster` name to the name of a real account. You can do this by editing the `aliases` file, which usually resides in `/etc` or sometimes in `/etc/mail`.

The `aliases` file format is fairly straightforward. Comment lines begin with hash marks (`#`), and other lines take the following form:

```
name: addr1[,addr2[,...]]
```

The *name* that leads the line is a local name, such as `postmaster`. Each address (*addr1*, *addr2*, and so on) can be a local account name to which the messages are forwarded, the name of a local file in which messages are stored (denoted by a leading slash), a command through which messages are piped (denoted by a leading vertical bar character), the name of a file whose contents are treated as a series of addresses (denoted by a leading `:include:` string), or a full e-mail address (such as `fred@example.com`).

A typical default configuration includes a few useful aliases for accounts such as `postmaster`. Most such configurations map most of these aliases to `root`. Reading mail as `root` is inadvisable, though—doing so increases the odds of a security breach or other problem because of a typo or bug in the mail reader. Thus, you may want to set up an alias line like the following:

```
root: yourusername
```

This redirects all of `root`'s mail, including mail directed to `root` via another alias, to *yourusername*, which can take any of the forms just described (it's most likely to be a local username or a valid remote e-mail address). Some mail servers, including `sendmail` and `Postfix`, require you to compile `/etc/aliases` into a binary file that can be processed more quickly. To do so, type **newaliases** as `root`.

Another approach to redirecting mail is to do so on the user level. In particular, you can edit the `~/ .forward` file in a user's home directory to have mail for that user sent to another address. Specifically, the `~/ .forward` file should contain the new address—either a username on the current computer or an entire e-mail address on another computer. This approach has the advantage that it can be employed by individual users—say, to consolidate e-mail from multiple systems into one account without bothering system administrators. A drawback is that it can't be used to set up aliases for nonexistent accounts or for accounts that lack home directories. The `~/ .forward` file can also be changed or deleted by the account owner, which might not be desirable if you want to enforce a forwarding rule that the user shouldn't be able to override.

Choosing a POP or IMAP Server

POP and IMAP are both tools that are used to deliver e-mail to end users, who typically use mail clients on remote computers. The two protocols differ in that POP is much simpler; it provides users with direct access to their mail queues on the server computer. Users may read and delete messages from that queue, but they may not organize their mail queues. To store messages in multiple folders according to content, correspondent, or other criteria, users are expected to download the mail and store it locally. IMAP, by contrast, supports the use of mail folders on the server computer. This feature enables users to access the same mail structure from several computers (say, a desktop system and a notebook). The drawback is that IMAP, if fully utilized, increases server requirements—mail stored long-term on the server increases disk needs, and users referring repeatedly to the same e-mail message will push up network bandwidth used by the server.



Several different versions of POP and IMAP exist. In particular, POP2 and POP3 are distinct protocols, and versions of both POP and IMAP that employ Secure Sockets Layer (SSL) encryption also exist.

Quite a few POP and IMAP servers are available for Linux. (Many programs handle both protocols.) Examples include the following:

UW IMAP Despite its name, the University of Washington IMAP server (<http://www.washington.edu/imap/>) supports POP2, POP3, and IMAP. This set of servers is extremely common; it ships with most Linux distributions, usually in a package called `imap` or `uw-imapd`. The IMAP server stores user mail folders in users' home directories, which can be awkward if users also log into their accounts and store nonmail files there.

Cyrus IMAP Like UW IMAP, Cyrus IMAP (<http://asg.web.cmu.edu/cyrus/imapd/>) supports more than just IMAP. Specifically, Cyrus IMAP supports IMAP, POP3, and a Kerberos-enabled POP3 variant (KPOP). This server stores IMAP mail folders in a proprietary file format in its own directory tree, so it can be a good choice if users store nonmail files in their home directories.

Courier The Courier mail server (<http://www.courier-mta.org>) is an integrated set of SMTP, POP, and IMAP servers. Although the Courier SMTP server isn't very popular in Linux, the IMAP server can be installed separately, and it has a modest following.

Qpopper This POP3 server was originally a commercial product released by the same people who developed the popular Eudora client and server packages. With version 4.0, though, Qpopper has gone open source. You can learn more at <http://www.eudora.com/products/unsupported/qpopper/>.

One issue you should consider when installing and configuring a pull mail server is password security. In a typical configuration, the traditional POP and IMAP servers require password authentication—users must enter their usernames and passwords. Most servers authenticate users against the normal Linux username and password database.

The basic protocols deliver the username and password over an unencrypted link. As a consequence, a miscreant with the appropriate access can sniff the password. The SSL variants fix this problem, but they require support in the mail clients. Another approach is to use the Secure Shell (SSH) to tunnel the pull mail protocol over an encrypted link, as outlined in Chapter 9. This approach requires configuring SSH on the server and on all the clients, though. If you don't want to go to this effort, you may want to consider setting aside special mail-only accounts and instruct users to create unique passwords for these accounts. This practice will at least minimize the damage that a miscreant might do if pull mail passwords are compromised. You may also want to restrict access to your POP or IMAP ports using firewall rules, TCP Wrappers, or `xinetd` access restrictions.

Actual POP and IMAP configuration varies substantially from one server to another. Given this fact, and the fact that this level of detail is not covered on the Linux+ exam, I recommend you consult server-specific documentation if you need to configure a POP or IMAP server.

Summary

Network servers are many and varied, and Linux provides programs to handle all common network protocols. These include DHCP for configuring clients' network settings, DNS for resolving hostnames and IP addresses, NTP for managing time on a network, POP and IMAP for pull e-mail, and SMTP for push e-mail.

Although these servers share certain common features, such as their ability to be launched via SysV startup scripts or super servers, each protocol and server is unique. Indeed, for many protocols, multiple servers are available, and each server can have its own unique configuration style. Thus, you must study each server you install and run to determine how best to manage it.

Exam Essentials

Summarize why you might want to run a DNS server. The Domain Name System (DNS) turns hostnames into IP addresses, and vice versa. You can run a DNS server both to handle this task for local computers (even those that aren't connected to the Internet) and to deliver this mapping to computers on the Internet at large so that outside users can access your local servers.

Explain the purpose of DNS zone files. DNS zone files contain the nitty-gritty details of DNS zones—that is, they hold mappings of IP addresses to hostnames, and vice versa. DNS zone files also hold miscellaneous extra zone information, such as the identities of mail servers and the e-mail address of the individual responsible for DNS zone maintenance.

Describe how you can maintain consistent times on all the computers on a network. The Network Time Protocol (NTP) handles this task. You run an NTP server that accesses time signals from one or more outside sources, thus synchronizing its clock to a (presumably accurate) clock. You can then configure all your network's other computers to synchronize their clocks with your first NTP server, thus keeping all your system's clocks closely synchronized with each other and almost as well synchronized with the outside source.

Summarize the common e-mail protocols. The Simple Mail Transfer Protocol (SMTP) is the protocol that's responsible for delivering mail between mail server computers on the Internet. It's also used by clients that initiate e-mail transfers. The Post Office Protocol (POP) and the Internet Message Access Protocol (IMAP) are two common protocols used by clients to retrieve e-mail messages. POP and IMAP are generally used in the last leg of e-mail delivery so that client systems can retrieve e-mail at their own convenience rather than receive it whenever the sender tries to deliver it, as SMTP works.

Explain how e-mail forwarding and aliases work. Ordinarily, a Linux mail server delivers mail to local accounts associated with the username in e-mail addressed to the server. Forwarding and aliases enable overriding this behavior so that mail addressed to nonexistent accounts can be delivered to real users or so that mail can be sent from a real recipient account to another account. Forwarding does this in a user-configurable way for one account, while aliases work system-wide but must be configured by the system administrator.

Review Questions

1. Which of the following protocols can be employed to use a third computer to authenticate users who want to log onto your computer remotely?
 - A. DNS
 - B. DHCP
 - C. NTP
 - D. Kerberos
2. Which of the following commands could you type to see whether the mail service is functioning and view a backlog of old messages?
 - A. **postfix**
 - B. **traceroute**
 - C. **sendmail**
 - D. **mailq**
3. Which protocols might an e-mail client use to retrieve e-mail from a mail server? (Choose all that apply.)
 - A. SMTP
 - B. POP3
 - C. MTA
 - D. IMAP
4. You've edited the `/etc/aliases` file on a mail server computer that runs sendmail. What command should you type to ensure that your changes take effect?
 - A. **newaliases**
 - B. **mailq --aliases**
 - C. **aliases --renew**
 - D. **sendmail -na**
5. Which of the following is an important distinguishing characteristic of Postfix vs. sendmail?
 - A. Postfix uses a more user-friendly configuration file format.
 - B. Postfix supports POP and IMAP as well as SMTP.
 - C. Postfix supports SMTP as well as POP and IMAP.
 - D. Postfix uses GPG to encrypt the e-mail it sends.

6. A university runs a mail server for its students and faculty, many of whom use public computers to log in and check their e-mail. Which of the following mail protocols will *best* enable users to organize and store their mail on the mail server so that it's accessible and properly organized when the users change from one client to another?
- A. SMTP
 - B. POP3
 - C. IMAP
 - D. KMail
7. A corporate mail server computer (`sendy.example.com`) runs sendmail on FreeBSD. An e-mail message sent through this computer is addressed to a user on a computer that runs Postfix on Linux (`posty.luna.edu`). Which of the following is a true statement?
- A. `sendy.example.com` will successfully send the e-mail to `posty.luna.edu`, assuming both are properly configured and there are no network problems.
 - B. `sendy.example.com` will be unable to connect to `posty.luna.edu` because the two servers require incompatible versions of SMTP.
 - C. The e-mail message will be successfully delivered, but only after a two-hour delay because of extensive protocol negotiations between sendmail and Postfix.
 - D. The e-mail message can be delivered successfully only if it's delivered via an intermediate system running Fetchmail, which bridges the differences between sendmail and Postfix.
8. A Linux system administrator types **`rndc flush`** at a **`root`** command prompt on a computer that's running a DNS server. What will be the effect?
- A. The DNS server program will restart.
 - B. All the DNS server's zone files will be emptied.
 - C. The DNS server's caches will be cleared.
 - D. The DNS server computer will restart.
9. What is the purpose of the following lines in `/etc/named.conf`?
- ```
zone "1.168.192.in-addr.arpa" {
 type master;
 file "named.192.168.1";
};
```
- A. They tell the DNS server to use the file `named.192.168.1` to look up IP addresses for names in the `in-addr.arpa` domain.
  - B. They tell the DNS server to use the file `named.192.168.1` to look up hostnames when given IP addresses.
  - C. They tell the DNS server to ignore ("zone out") requests in the `192.168.1.x` IP address block.
  - D. They tell the DNS server to retrieve the file `named.192.168.1` from the master server for the domain in question.

10. In which directory are DNS zone files typically stored on a system that runs the main Linux DNS server, `named`?
- A. `/etc/dns`
  - B. `/var/dns`
  - C. `/etc/named`
  - D. `/var/named`
11. Why might you run a DNS server? (Choose all that apply.)
- A. To deliver IP addresses for local users
  - B. To deliver IP addresses for off-site users
  - C. To deliver IP addresses for NetBIOS users
  - D. To deliver IP addresses for IPX users
12. Which of the following are legal record types in a DNS zone file? (Choose all that apply.)
- A. MX
  - B. DNS
  - C. ZF
  - D. CNAME
13. Which of the following options is true of the following DNS zone file entry?

```
example.net. IN SOA dns.pangaea.edu. \
 fred.example.com. (
 7 ; serial
 3600 ; refresh
 600 ; retry
 604800 ; expire
 86400 ; default_ttl
)
```

- A. The serial number (7) is invalid; this number must be a date-based code, such as 2010071101.
- B. The primary DNS server entry (`dns.pangaea.edu`) is invalid; this server must exist within the main (`example.net`) domain.
- C. You should send e-mail to `fred@example.com` concerning any DNS-related problems with the `example.net` domain.
- D. This domain has precisely two DNS servers: `dns.pangaea.edu` and `fred.example.com`.

14. You want to enable users to access the computer with the IP address of 192.168.17.198 as `linus.example.com`. What line would you place in the zone file for `example.com` to accomplish this task?
- A. `linus IN A 192.168.17.198`
  - B. `linus IN MX 192.168.17.198`
  - C. `198 IN TXT linus.example.com.`
  - D. `198 IN PTR linus.example.com.`
15. What is the effect of the following two DNS zone file entries for the `luna.edu` domain?
- ```
tycho.luna.edu.      IN A      192.168.23.5
www                 IN CNAME  tycho
```
- A. The same computer (192.168.23.5) may be accessed as either `tycho.luna.edu` or `www.tycho.luna.edu`.
 - B. The same computer (192.168.23.5) may be accessed as either `tycho.luna.edu` or `www.luna.edu`.
 - C. E-mail sent to `www.luna.edu` is delivered to `tycho.luna.edu`.
 - D. The server will fail to start, since a stray dot (.) appears at the end of the hostname `tycho.luna.edu`.
16. A network has two servers with static IP addresses, ten desktop systems, and five laptop computers that come and go. The desktop and laptop systems are configured to use DHCP to obtain their IP addresses. Which of the following is the *most* reasonable number of IP addresses to reserve for this network in the DHCP server's configuration?
- A. 10
 - B. 15
 - C. 100
 - D. Unlimited
17. You want your DHCP server to deliver your network's DNS servers' IP addresses to clients. Which of the following lines, in `/etc/dhcpd.conf`, will accomplish this goal?
- A. `set dns 192.168.1.7, 10.10.7.192`
 - B. `option domain-name-servers 192.168.1.7, 10.10.7.192;`
 - C. `option netbios-name-servers 192.168.1.7, 10.10.7.192;`
 - D. `set domain 192.168.1.7 10.10.7.192;`
18. Which of the following systems' IP addresses should you never configure via a DHCP server?
- A. The IP addresses used by Windows workstations
 - B. The IP address used by the DHCP server computer itself
 - C. The IP addresses used by SMTP mail server computers
 - D. The IP addresses used by network-enabled printers

19. What will be the effect of a computer having the following two lines in `/etc/ntp.conf`?

```
server pool.ntp.org
server tardis.example.org
```

- A. The local computer's NTP server will poll a server in the public NTP server pool; the first `server` option overrides subsequent `server` options.
 - B. The local computer's NTP server will poll the `tardis.example.org` time server; the last `server` option overrides earlier server options.
 - C. The local computer's NTP server will poll both a server in the public NTP server pool and `tardis.example.org` and use whichever site provides the cleanest time data.
 - D. The local computer's NTP server will refuse to run because of a malformed `server` specification in `/etc/ntp.conf`.
20. You've configured one computer on your five-computer network, `gateway.pangaea.edu`, as an NTP server that obtains its time signal from `ntp.example.com`. This configuration provides an acceptably accurate clock on `gateway.pangaea.edu`. What computer(s) should your network's other computers use as their time source(s)?
- A. You should consult a public NTP server list to locate the best server for you.
 - B. Both `gateway.pangaea.edu` and `ntp.example.com`.
 - C. Only `ntp.example.com`.
 - D. Only `gateway.pangaea.edu`.

Answers to Review Questions

1. D. Kerberos is a network authentication tool; clients and servers that are written to use Kerberos can consult with the Kerberos server to perform authentication tasks. DNS is the Domain Name System; it translates between IP addresses and hostnames. DHCP is the Dynamic Host Configuration Protocol; it delivers IP addresses to clients for basic network setup. NTP is the Network Time Protocol; it keeps computers' clocks synchronized. None of these three protocols is used for network authentication, so options A, B, and C are all incorrect.
2. D. The `mailq` utility can display a backlog of old messages and show you whether the mail service is functioning. Postfix and sendmail are both mail server programs (although typing **sendmail -bp** is equivalent to typing **mailq**), while `traceroute` is an enhanced version of `ping` that shows the route data takes to reach a target.
3. B, D. The Post Office Protocol version 3 (POP3) and the Internet Message Access Protocol (IMAP) are two common protocols used for retrieving e-mail from the final mail servers in the mail delivery chain. The Simple Mail Transfer Protocol (SMTP) is used for sending e-mail, both by the originating system and by mail servers between the originator and the final recipient, but it's not used by e-mail clients to retrieve e-mail from the server. A mail transfer agent (MTA) is another name for an e-mail server program; it's not an e-mail protocol per se.
4. A. The `newaliases` command processes the text-mode `/etc/aliases` file into a binary form that the mail server can process more rapidly. This command must be run for your changes to take effect. Options B and C are both fictitious options to real commands, and option C is a fictitious command.
5. A. Postfix's configuration file format is easier for nonexperts to understand and manipulate than is sendmail's configuration file format, which is notoriously opaque. Both Postfix and sendmail support SMTP, and neither supports POP or IMAP (although you can run separate POP or IMAP servers alongside either Postfix or sendmail). Neither Postfix nor sendmail directly supports GPG, but you can send GPG-encrypted messages using either mail server by using a mail client or other software to encrypt the message.
6. C. The Internet Message Access Protocol (IMAP) enables users to permanently store and organize e-mail on the server, as specified by the question. The Simple Mail Transfer Protocol (SMTP) is a push mail protocol that's not used for mail retrieval by end users. The Post Office Protocol version 3 (POP3) is a pull mail protocol, but it doesn't provide options for organizing mail on the server, so it won't work well for the stated purpose. KMail is one of many mail clients for Linux; it's not a mail protocol at all, although it supports SMTP, POP3, and IMAP.
7. A. Sendmail and Postfix both understand the same Simple Mail Transfer Protocol (SMTP) and can therefore send and receive e-mail from each other with no problems, assuming proper configuration; thus, option A is correct. Contrary to options B, C, and D, there are no fundamental incompatibilities that will block e-mail delivery, delay e-mail delivery, or require an intermediate system. (That said, network problems, system load issues, and so on, can unpredictably delay e-mail delivery in the real world, but such delays aren't the result of sendmail/Postfix incompatibilities.)

8. C. The `rndc` program provides a control interface for the DNS server program, `named`. The `flush` option to `rndc` clears (flushes) the cache of recent DNS lookups, so option C is correct. None of the remaining options describes anything that `rndc` can do, although the `reload` option will cause `named` to reload its zone files, which is partway to option A.
9. B. Zone file definitions in `/etc/named.conf` tell the DNS server where to look to find mappings of IP addresses to hostnames, or vice versa. The `in-addr.arpa` pseudo-domain is reserved for reverse DNS lookups—the server returns hostnames when given IP addresses. The `file` directive points the server to the file that holds the mappings. Thus, option B is correct. Option A is incorrect because reverse DNS lookups return hostnames, not IP addresses. A `zone` directive is not an instruction to ignore a domain or address block, contrary to option C. The `type master` line in the example tells the server that it is the master DNS server for this zone, not to retrieve a file from another master server, so option D is incorrect.
10. D. The `/var/named` directory holds DNS zone files on a typical Linux DNS server that runs `named`. Options A, B, and C are all fictitious; these locations don't normally exist.
11. A, B. A DNS server can deliver IP addresses for local users, off-site users, or both, depending on its and your network's configuration; thus, options A and B are both correct. NetBIOS and IPX are alternative network stacks, neither of which uses the TCP/IP addresses that DNS delivers. Thus, options C and D are both incorrect.
12. A, D. Mail exchanger (MX) and canonical name (CNAME) records are common in forward zone files. There are no such things as DNS and ZF records. Other common record types include Address (A), Name Server (NS), Pointer (PTR), and Start of Authority (SOA) records. (PTR records are used in reverse zone files.)
13. C. Option C is correct; an administrative contact e-mail address is embedded in the SOA record, but with the at-sign (@) replaced by a dot (.). Option A is incorrect because the serial number need not be date-based, although date-based serial numbers are common. Option B is incorrect because the primary DNS server for a domain need not be part of the domain it serves. Option D is incorrect because the SOA record identifies only one DNS server; others may be specified via separate NS records.
14. A. The question is asking for an address (A) record entry for the forward zone file, and option A presents such an entry. Option B presents a valid mail exchanger (MX) record, but that's not what the question asked for. Option C presents a text (TXT) record that might appear in a reverse zone file, but it's a somewhat odd one. Option D presents a correct pointer (PTR) record for the reverse zone file, but that's not what the question asked for.
15. B. A canonical name (CNAME) record creates a sort of alias, enabling a second name to access a computer that already has a hostname. Option B correctly describes the effect of the CNAME record in this example, given the address (A) record that also appears in this example. The CNAME entry shown does *not* create a name of `www.tycho.luna.edu`, contrary to option A. E-mail delivery to a domain can be adjusted via an MX record, but the example includes no such record; and even if the second record were an MX record rather than a CNAME record, option C would not describe its effect. The dot at the end of `tycho.luna.edu` in the first entry identifies a complete hostname; it's either required, or the domain name portion of the name must be omitted (as in `www` on the second line), contrary to option D.

16. C. When configuring a DHCP server's address space, you should give it plenty of slack, both to allow for network expansion and to permit clients to obtain new addresses in case a lease isn't released. Option A's 10 addresses isn't enough for the 15 desktop and laptop systems that are expected to use the DHCP server, and option B's 15 addresses are barely sufficient—any unexpected demands will cause clients to receive no addresses. Option C's 100 provides plenty of room for expansion or leases temporarily lost because of clients that don't release their leases. Option D isn't a possibility, since DHCP requires you to explicitly set aside a finite set of addresses.
17. B. The option `domain-name-servers` line sets the DNS server IP addresses delivered by the DHCP server to its clients, and option B uses the correct syntax for this option. Options A and D are both incorrect because the `set` keyword isn't used for this purpose in DHCP. Option C specifies the `netbios-name-servers` feature, which refers to NetBIOS name servers (used by SMB/CIFS), not DNS servers.
18. B. The DHCP server computer must be configured with an IP address before it can work, so you can't use a DHCP server to give itself an IP address; the DHCP server computer must have a static IP address, making option B correct. Windows workstations and network servers (including network-enabled printers) can obtain their IP addresses via DHCP, although if the server requires a fixed IP address, you'll have to perform more advanced DHCP configuration than is described in this chapter.
19. C. Multiple `server` entries in `/etc/ntp.conf` tell the system to poll all the named servers and to use whichever one provides the best time data. Thus, option C is correct. (The `pool.ntp.org` subdomain and numbered computers within that subdomain give round-robin access to a variety of public time servers.) Options A and B both incorrectly state that one `server` statement overrides another, when in fact this isn't the case. The `server` statements shown in the question are properly formed.
20. D. Once you've configured one computer on your network to use an outside time source and run NTP, the rest of your computers should use the first computer as their time reference. This practice reduces the load on the external time servers, as well as your own external network traffic. Thus, option D is correct. (Very large networks might configure two or three internal time servers that refer to outside servers for redundancy, but this isn't necessary for the small network described in the question.) Option A describes the procedure to locate a time server for the first computer configured (`gateway.pangaea.edu`) but not for subsequent computers. Although configuring other computers to use `ntp.example.com` instead of or in addition to `gateway.pangaea.edu` is possible, doing so will needlessly increase your network traffic and the load on the `ntp.example.com` server.



Chapter 11

Configuring Network Servers II

THE FOLLOWING COMPTIA OBJECTIVES ARE COVERED IN THIS CHAPTER:

- ✓ 2.7 Manage filesystems using the following (NFS: configuration, exportfs, /etc/exports, showmount).
- ✓ 3.2 Implement interoperability with Windows using the following (rdesktop—client, vnc—server and client, Samba—server and client [smb.conf, winbind, lmhosts]).
- ✓ 3.3 Implement, configure, and maintain Web and FTP services (Apache, FTP services).
- ✓ 3.4 Given a scenario, explain the purpose of the following web-related services (Tomcat, Apache, Squid).
- ✓ 3.5 Troubleshoot web-related services using the following utilities (Commands: curl, wget, ftp, telnet).
- ✓ 3.6 Given a scenario, troubleshoot common FTP problems (Active vs. passive, ASCII vs. binary).
- ✓ 3.7 Given a scenario, perform the following MySQL administrative tasks (Locate configuration file, Starting and stopping, Test the configuration).



Chapter 10, “Configuring Network Servers I,” began an exploration of Linux network servers. This chapter continues this investigation with a look at several more servers. Specifically, this chapter covers several types of file sharing and exchange servers (the Network File System, Samba, the File Transfer Protocol, and the Apache Web server), Windows remote-access tools (rdesktop and Virtual Network Computing), and the Structured Query Language database tool.



Objective 2.7 is covered partly in this chapter and partly in Chapter 6, “Managing Disks.” Objective 3.2 is covered partly in this chapter and partly in Chapter 10.

Delivering Files Over the Network

Many network protocols involve file transfer in one form or another. This is most directly the case with file-sharing and file-transfer protocols, such as the Server Message Block/Common Internet File System (SMB/CIFS), the Network File System (NFS), and the File Transfer Protocol (FTP). The World Wide Web’s (WWW’s) protocol, the Hypertext Transfer Protocol (HTTP), is another common protocol for transferring files. Naturally, Linux can function as a server for any of these protocols.



As I use the term, a “file-sharing protocol” is used to share files in a way that makes it easy for a client to mount the shared-file directory as if it were a local partition. SMB/CIFS and NFS are both file-sharing protocols. File-transfer protocols, on the other hand, lack some low-level features that are helpful for file sharing and so are seldom used for this purpose. Instead, file-transfer protocols, such as FTP and HTTP, are generally used to transfer entire files with the help of specialized programs, such as the ftp client program or Web browsers.

Delivering Files with Samba

The Samba server suite (<http://www.samba.org>) implements the SMB/CIFS protocol suite, which is used for sharing files and printers. It’s controlled through a master configuration

file, `smb.conf`, which is usually stored in `/etc/samba` (it can sometimes appear in `/etc`, `/etc/samba.d`, or other locations). The Samba suite actually consists of two major servers, `smbd` and `nmbd`, along with several auxiliary programs and servers. It's started via one or more SysV startup scripts. Typically, either one script called `samba` (or something similar) starts both `smbd` and `nmbd` or these servers have independent SysV startup scripts named after themselves.



Non-Windows computers, including Linux, can function as Samba clients. Chapter 6 describes using Linux as an SMB/CIFS client.

Setting Basic Samba Options

The main `smb.conf` file consists of several sections, each of which begins with a keyword in square brackets. The first of these is the `[global]` section, which sets global options. Subsequent sections define file or printer *shares*—named resources on the SMB/CIFS server that can be accessed to share files or printers. Each of these sections is named with a share name; for instance, `[common]` defines a share called `COMMON`. Within each section, Samba parameters look like this:

parameter = *value*

parameter is a keyword, such as `security` or `netbios name`. It's essentially a variable name that's set to the *value* specified. This *value* may be numeric (such as a time in seconds), a filename, a hostname, a Boolean (Yes or No; True or False; 1 or 0), or some other type of value. The intent is that the `smb.conf` file's meaning be fairly self-explanatory, at least if you know the server's basic features. Unfortunately, there are so many features that you might not understand everything, much less be able to generate new entries, unless you're an expert. Most default `smb.conf` files include extensive comments to help you with this process, or you can consult the `smb.conf` man page, which is unusually complete.

Two parameters in the `[global]` section are particularly important: `workgroup` and `encrypt passwords`. The `workgroup` parameter sets the name of the NetBIOS workgroup or domain. The default value is usually `WORKGROUP`, but most networks have their own workgroup name, so you should adjust this parameter appropriately. If you don't, clients may have trouble finding the Samba server using their network browsers. The `encrypt passwords` parameter is a Boolean that determines whether Samba uses its own encrypted password database or requires unencrypted passwords that it authenticates using the standard Linux password database. The default value is `No` for Samba 2.x but is `Yes` for Samba 3.0 and later. In most cases, `encrypt passwords = Yes` is the most appropriate choice, because all versions of Windows since the mid-1990s require the use of encrypted passwords by default.

Unfortunately, using encrypted passwords means that you must maintain a Samba-specific password database. The reason is that the encrypted password exchange tools of SMB/CIFS are incompatible with the methods Linux uses to store passwords in `/etc/shadow`. To create and maintain an encrypted password database, you can use the `smbpasswd` utility, which stores data in the `/etc/samba/smbpasswd` file by default. (This file sometimes resides elsewhere,

particularly if you compile Samba yourself. You can also use the `passdb` backend option to alter how Samba stores its encrypted passwords; consult the `smb.conf` man page for details.) To add a user to the Samba password database, pass `smbpasswd` the `-a` parameter and the username:

```
# smbpasswd -a john
```

This command creates an entry for `john`, provided that the local Linux user `john` already exists. (The `smbpasswd` program will create entries only for those users who already have standard Linux accounts.) When you type this command, you'll be prompted twice for a password, much as when you use the Linux `passwd` command to change a password. In fact, you can then use `smbpasswd` to change Samba passwords for existing users.



The first time you use the `smbpasswd` command, it will complain about the lack of an `/etc/samba/smbpasswd` file. This complaint looks like an error message, but the utility creates the file, so nothing is wrong or needs your attention.

Resolving Hostnames in Samba

SMB/CIFS was originally built around non-TCP/IP networking, but today's implementations generally use TCP/IP. The result can be a bit confusing when it comes to name resolution, since implementations differ in how they resolve names—some use TCP/IP hostnames, but others use NetBIOS names, even when working over TCP/IP.

To maximize flexibility, Samba supports a file called `lmhosts`, which is typically stored in `/etc/samba`. This file is similar to the Unix `/etc/hosts` file, in that it provides a mapping of hostnames to IP addresses. It consists of one line per host, with an IP address and hostname:

```
172.24.21.34 DANCE
```

This line maps the NetBIOS name `DANCE` to `172.24.21.34`. If you have problems locating Windows servers using Samba clients, editing `lmhosts` may fix the problem. Sometimes, though, this file will be useless. In particular, if your servers' IP addresses change frequently (say, because you're using DHCP with dynamic IP addresses to configure them), then a static `lmhosts` file will go quickly out-of-date. In that case, the standard NetBIOS name resolution should be used. Samba tries to do this by default, but to be sure, check the `[global]` section of your `smb.conf` file for the `name resolve order` directive:

```
name resolve order = host lmhosts wins bcast
```

Each subsystem is tried in the order listed. If a subsystem is omitted from the list, it's not used. In this example, `host` uses Linux's standard DNS lookup, `lmhosts` uses the `lmhosts` file, `wins` uses a NetBIOS equivalent to a DNS server, and `bcast` uses low-level broadcast lookups that are unique to SMB/CIFS. Try different orders of these four options until you find a combination that works.

Sharing Files with Samba

To share files with Samba, you must create file share definitions. These can be as simple as a share name in square brackets in the `smb.conf` file:

```
[sample]
```

This line creates a share called `SAMPLE`. If no other lines are present until the end of the file or the next share definition, the share is read-only and provides access to the `/tmp` directory. To make the share useful, chances are you'll want to change at least some of these defaults:

```
[sample]
```

```
comment = Sample Samba Share
path = /home/samba/sample
read only = No
```

This example sets three parameters. The `comment` line sets a comment string that's associated with the share and that appears in many clients' network browsers. It doesn't directly affect the share's functionality, but providing a descriptive comment can help users find the shares they need. The `path` parameter tells Samba what directory to share. The default value is `/tmp`, but that's not usually very useful. A synonym for this parameter is `directory`. Finally, setting `read only = No` tells Samba that users may write to the share. The `writable`, `writable`, and `write ok` parameters are antonyms for `read only`; that is, `writable = Yes` is equivalent to `read only = No`. An important caveat about Samba's write permissions is that they still work within the constraints of Linux file permissions, as described in Chapter 2. Every user who accesses a Samba server does so as an ordinary user. If that user can't write to a directory or file, Samba won't permit the user to do so (at least, not without using more advanced parameters). Thus, you must ensure that permissions are set appropriately within file shares if you want your users to be able to write to them. In fact, permissions must be set to enable users to *read* files in shares that they should be able to read, as well. In most cases, `0644 (-rw-r--r--)` permissions do nicely for files in read-only shares, but managing permissions in read/write shares can be complex.

An important special Samba share is the `[homes]` share. Unlike most shares, this one doesn't point to a single directory; it points to the user's home directory, as defined in `/etc/passwd`. If you want users to be able to store their personal files on a Samba server, a `[homes]` share can be just the thing you need. Most `smb.conf` sample files include a working `[homes]` share, so you might not need to do anything to add one. In operation, this share appears as the user's username—for instance, the user `john` sees a share called `JOHN`.



Samba is a very complex and powerful server, and this description barely presents the most basic Samba information. If you need to do more with Samba, you should consult its copious documentation or a book on the subject, such as my *Linux Samba Server Administration* (Sybex, 2001).

Sharing Printers with Samba

In addition to file shares, Samba supports printer shares. The simplest way to share printers is with a `[printers]` share, which is analogous to a `[homes]` share in that both share every available instance of their type (printers or user home directories, respectively). A typical `[printers]` share definition looks like this:

```
[printers]
    comment = "All Printers"
    path = /var/spool/samba
    print ok = Yes
```

A share called `[printers]` causes all the printers defined on the Samba server to be shared with SMB/CIFS clients. This definition depends on Samba to be able to identify the printers. On a Linux system that uses the Common Unix Printing System (CUPS) for printing, you should normally include a line that reads `printing = cups` in the `[global]` section in order to tell Samba how to interface with the printing system.

Instead of, or in addition to, a `[printers]` definition, you can create named printer-by-printer definitions. These, like the `[printers]` share, must include the `print ok = Yes` parameter or its synonym, `printable = Yes`. By default, Samba looks for a local printer with the same name as the printer share; however, you can point Samba to a different share by using the `printer name` parameter, as in `printer name = laser` to print to a local queue called `laser`.

Print jobs accepted by Samba are submitted to the Linux printing system as if they were locally generated. In most cases, this means that clients should be configured as if they were printing to a PostScript printer, even when the printer isn't a PostScript model. Generic PostScript drivers usually work well, as do drivers for Apple LaserWriter printers. For color printers, try using a driver for a QMS magicolor printer; these drivers usually work well with Samba. Some printer queues' smart filters recognize native printer languages and pass such files through unmodified. If this is the case, using the drivers provided by the printer manufacturer will work. Ultimately, you may need to experiment to determine what works well on the client side.

Using a Windows Domain

Most networks that rely heavily on SMB/CIFS for file sharing deploy a domain configuration, which differs from the simpler workgroup configuration in several ways. The most important of these is that a domain includes a *domain controller* computer, which manages accounts for all the computers on the domain.

If you want to use a Windows domain with your Samba server (or Linux SMB/CIFS clients), you must set a couple of `smb.conf` global options:

```
security = Domain
password server = 172.24.21.98
```

The `security` option tells Samba and related tools how to handle authentication. The default value, `User`, tells Samba to authenticate users against a local account database, as outlined in “Setting Basic Samba Options.” Three different options may be used on a Windows domain:

Server Setting `security = Server` causes Samba to use the Windows NT 4 protocols to talk to a domain controller, but without fully joining the domain. This configuration is easy to set up, but the Samba developers discourage its use.

Domain Setting `security = Domain` causes Samba to use the Windows NT 4 protocols to talk to a domain controller as a full domain member. Prior to accessing the controller for the first time, you must join the domain by typing `net join member -U adminuser`, where `adminuser` is an account on the domain controller that has administrative access to the domain controller’s account database in order to create a special account for the Linux system. You’ll be asked to type a password, and if all goes well, you’ll then see a message to the effect that you’ve joined the domain.

ADS Setting `security = ADS` works much like `security = Domain`, except that the system uses the more sophisticated Active Directory (AD) protocols rather than the older NT 4 protocols. This configuration can be finicky in Linux, however.

With the `security` option set, the `password` server pointing to the domain controller computer (you can specify an IP address or a NetBIOS name), and, if necessary, a domain account created for the computer, Samba should now defer to the domain controller for authentication tasks.

If authentication succeeds but no local Linux account exists for a user, the user won’t be able to log in. If you want the existence and successful authentication of a user on a domain to be sufficient, you can employ the `add user` script Samba option:

```
add user script = /usr/sbin/useradd -m %u
```

This configuration causes Samba to run `useradd` with the `-m` option (to create a user home directory) when a domain login succeeds but no local user exists. (The `%u` option is a Samba variable that refers to the username.)

Many Linux systems use Windows domains solely for Samba authentication. It’s possible, though, to employ Windows domains in a more comprehensive manner. The `Winbind` tool (sometimes referred to as `winbindd`, the name of the server binary) can link Windows domain accounts into the main Linux account system. Once so configured, the computer will use the Windows NT 4 domain or AD controller to authenticate users for ordinary text-mode and GUI logins, e-mail retrieval, FTP accesses, and so on. Configuring `Winbind` can be tricky, since it requires setting options in the `smb.conf` file, in the `/etc/nsswitch.conf` file, and in one or more files in `/etc/pam.d`. Getting it wrong can result in a system that doesn’t accept any logins. Chapter 12, “Securing Linux,” describes these authentication tools in more detail, but if you want to configure `Winbind`, you may want to consult more advanced documentation, such as a book on Samba or network authentication.

Delivering Files with NFS

Samba is primarily a tool for file sharing with Windows clients. Although it can be used for sharing files with Linux or Unix clients, SMB/CIFS wasn't designed with these systems in mind. Early versions of SMB/CIFS lack support for Unix-style ownership and permissions. Later versions add this functionality, although it frequently works imperfectly because the Windows and Unix ownership and permissions models tend to conflict with each other. A better option for file sharing between Linux and Unix systems is the *Network File System (NFS)*, which was designed by Sun as a network file-sharing tool for Unix. To use NFS, you must configure the directories you're sharing (known as *exports* in NFS) and start the server. You can also use utilities to modify or review the configuration once the server is running.

Configuring the NFS Server

In Linux, NFS server configuration is handled through a file called `/etc/exports`. This file contains lines that begin with a directory that's to be shared followed by a list of hostnames or IP addresses that may access it, with their options in parentheses:

```
/home taurus(rw,async) littrow(ro)
/opt taurus(ro)
```

These examples share two directories: `/home` and `/opt`. Two computers (`taurus` and `littrow`) may access `/home`, but only `taurus` may write to that directory because only `taurus`'s definition includes the `rw` (read/write) option; the `littrow` definition includes the `ro` (read-only) specification. The `/home` description for `taurus` also includes the `async` option, which can improve performance but slightly increases the risk of data loss should a disk error occur. The `/opt` directory is shared only with `taurus`, and that system may not write to the directory.

In order to deliver NFS support, you must run an NFS server program. (Most NFS server programs for Linux rely on special kernel features as well, but they are almost always compiled into the kernel by default.) Typically, this program is run by a SysV startup script, often called `nfsserver` or something similar. Check for this startup script and, if necessary, start or restart it once you've made changes to the `/etc/exports` file.

Modifying or Viewing the NFS Configuration on the Fly

In addition to `/etc/exports`, NFS enables nonpermanent changes to its exports via the `exportfs` command. Used without any options, `exportfs` displays a list of active NFS exports, similar to the contents of `/etc/exports`, but with one line per export (so if you list a directory as being exported to three systems or networks, three lines will appear in the `exportfs` output for that directory). Adding options enables you to modify your system's NFS exports. Table 11.1 summarizes the most important of these options. The `exportfs` man page provides details on more obscure options.

TABLE 11.1 Common `exportfs` Options

| Option | Explanation |
|-------------------------|---|
| <code>-a</code> | Reads <code>/etc/exports</code> and exports all the directories listed there. (When used with <code>-u</code> , unexports all directories.) |
| <code>-r</code> | Reexports all directories. This has the effect of unexporting directories that are not listed in <code>/etc/exports</code> . |
| <code>-o options</code> | Implements the specified <i>options</i> , which take the same form as those in <code>/etc/exports</code> . |
| <code>-u</code> | Unexports one or more directories. |
| <code>-f</code> | Flushes and rebuilds the exports table. |
| <code>-v</code> | Adds verbose messages to the program's output. |

When using `exportfs` to add or delete exports, you specify a client and directory in the form *client:/export/directory*. You don't need to specify any of the options from Table 11.1 when exporting a new directory, but you must use the `-u` option to unexport a directory. For instance, suppose your NFS server is currently exporting `/var/www` to `192.168.23.0/24` as a means to enable local users to edit a Web server's files. You want to move this directory to `/var/apache/webfiles`. You could implement these changes by typing the following commands:

```
# exportfs -u 192.168.23.0/24:/var/www
# exportfs 192.168.23.0/24:/var/apache/webfiles
```

This change will be temporary, however; you should also edit `/etc/exports`. In fact, you might prefer to edit `/etc/exports` first and then type **`exportfs -r`**, thus implementing your changes.

A second tool for managing NFS is `showmount`, which displays information on current NFS activity. Used without options, this tool reveals the IP addresses of the computers that are currently using the server. Table 11.2 summarizes `showmount`'s most common options.

TABLE 11.2 Common `showmount` Options

| Option | Option abbreviation | Explanation |
|----------------------------|---------------------|--|
| <code>--all</code> | <code>-a</code> | Displays both the IP addresses of clients and the directories they're using |
| <code>--directories</code> | <code>-d</code> | Displays the directories currently being shared by the server, but not the identities of clients |

TABLE 11.2 Common showmount Options (*continued*)

| Option | Option abbreviation | Explanation |
|--------------|---------------------|--|
| --exports | -e | Displays the current available exports (similar to the default output of <code>exportfs</code> , but each export uses just one line of output) |
| --help | -h | Presents basic help information |
| --version | -v | Prints the program's version number |
| --no-headers | None | Suppresses explanatory headers in the output |

You can use `showmount` to display information on the server running on any computer (network firewall and other security options permitting). By default, the program displays information on the local computer's NFS server; but if you add a computer hostname, the result is information on that computer:

```
$ showmount -a nessus
All mount points on nessus:
172.24.21.5:/home
172.24.21.5:/home/george/photos
192.168.1.4:/home
192.168.1.4:/home/sally
```

In this example, 172.24.21.5 is accessing `/home/george/photos` (part of the `/home` export), and 192.168.1.4 is accessing `/home/sally` (also part of the `/home` export).

Understanding NFS Security Concerns

Most servers use passwords or some other authentication tool to control access to files. NFS works differently; an NFS server trusts the client system to control access to files. Once a directory is exported via NFS, any client computer that's authorized to access the directory in `/etc/exports` may do so in any way the `/etc/exports` definition permits. The idea is that the client computer will have a user base that's compatible with the user base on the server and that the client computer is trustworthy.

These assumptions weren't unreasonable when NFS was created, but in today's computing environment, they're a bit risky. Somebody with a notebook computer and wireless networking hardware may be able to access your server and masquerade as another computer if you use a wireless network. Even with a wired network, a compromised system or physical access can enable an attacker to pretend to be a trusted system. An attacker can control the user database on the attacking computer or use a custom NFS client program that doesn't play by the usual security rules, thus bypassing the intent of the NFS security scheme. Therefore,

you should be cautious about NFS security. Don't add a computer to `/etc/exports` unless it's really necessary, and don't give clients read/write access unless they really need it. You might also want to use IP addresses rather than hostnames to specify computers in `/etc/exports`; this practice makes masquerading as a trusted system a little more difficult.

Delivering Files with FTP

FTP has long been a popular server, providing a login-based means of retrieving files. The protocol has some peculiarities, but every OS that has a serious TCP/IP stack has an FTP client. FTP is typically used in one or both of two ways:

- Users must authenticate themselves to the server by providing a username and password. They can then read, and often write, files to their home directory or to common areas on the computer.
- Users provide a username of `anonymous` and any password (conventionally their e-mail addresses). They can then read, but usually not write, data stored in public directories. This *anonymous FTP* access is a popular means of delivering public files such as software upgrades, multimedia files, and so on.

Both configurations share many features, but certain details differ. How you set up an FTP server to use either system depends on the server you choose. Several such servers exist for Linux. This section describes your choices and then covers two popular FTP servers, ProFTPD and vsftpd, in more detail. I also describe Linux FTP clients and some common FTP pitfalls.



One of FTP's major problems when used for authenticated user access is that FTP sends all data, including passwords, in an unencrypted form. This fact means that miscreants on the server's network, the client's network, or intervening networks can use packet sniffers to steal users' passwords. This issue isn't as much of a problem for anonymous access, which is supposed to be public.

Choosing an FTP Server

FTP is an old protocol, and numerous implementations of it have sprung up over the years. These servers vary in many details; however, they all serve the same protocol, and they all look very much alike to their users. FTP server options for Linux include the following:

ProFTPD This server, headquartered at <http://proftpd.org>, is one of the more popular of the very complex FTP servers. It ships with most major Linux distributions. Its configuration file is modeled after that of Apache, and the server supports many advanced features. A dedicated GUI configuration tool, GProFTPD (<http://freshmeat.net/projects/gproftpd/>), is available for this server.

vsftpd This server aims to excel at security, stability, and speed. In doing so, its developers have chosen to eschew some of the more advanced features of servers such as ProFTPD and WU-FTPd. If you don't need those features, this trade-off may be more than acceptable. You can learn more from its Web site, <http://vsftpd.beasts.org>. It's available with most Linux distributions.

WU-FTPd The Washington University FTP Daemon (WU-FTPd) is an old standard in the Linux world. Unfortunately, it has collected more than its fair share of security problems and isn't the speediest FTP server available. For these reasons, it ships with fewer Linux distributions today than in years past. Its main Web site is <http://www.wu-ftp.org>.

PureFTPd This server, headquartered at <http://www.pureftpd.org>, is another FTP server that emphasizes security.

oftpd This server is unusual because it's designed to function *only* as an anonymous FTP server; it doesn't support logins using ordinary user accounts. This feature can be appealing if you only want to run an anonymous server, but it makes this server unsuitable for many other purposes. It's available from <http://www.time-travellers.org/oftpd/>.



This list of FTP servers is far from complete. The oftpd Web page describes the primary developer's search for a name for the server. He wanted a name of the form xftpd, where x was a single letter. Starting with a, he found that oftpd was the first name that wasn't already in use!

Because FTP can potentially provide users with substantial access to the system—the ability to read or write any file, within limits imposed by Linux file ownership and permissions—FTP servers are unusually sensitive from a security point of view. As a result, the Web pages for many of the servers in the preceding list emphasize their developers' attention to security.

For a small FTP site, chances are any of the servers in the preceding list will work well, with the exception of oftpd if you want authenticated user logins rather than anonymous access. ProFTPD and vsftpd are both popular choices on modern Linux systems.

FTP Server Configuration

Details of FTP server configuration vary greatly from one server to another, so I don't describe most features in detail. Broadly speaking, though, you should pay attention to several features:

Configuration files Most FTP servers have configuration files, such as `/etc/proftpd/proftpd.conf` for ProFTPD or `/etc/vsftpd/vsftpd.conf` for vsftpd. You should consult your server's documentation to learn the details of the configuration file format and the options that the server supports.

Launch method Many FTP servers can run either stand-alone (via a SysV or local startup script) or from a super server. Some servers work better when launched in one way or the other, and distributions sometimes provide startup scripts for one method but not the other. Chapter 4, "Managing System Services," describes both methods of launching servers, so you should consult it and check to see what startup scripts are provided with your package.

chroot jails FTP servers often run in a *chroot jail*, which is a way to run a program such that it can see only a limited subset of the computer's available directories. You should consult your server's documentation to learn whether it supports this feature and, if so, how to use it. Typically you'll set aside a directory and copy a few files to it, including the files that the FTP server should serve.

User configuration FTP servers frequently enable any user with a regular account to log in. Some servers also support the `/etc/ftpusers` file, which includes the usernames of users who may *not* use the FTP server. This file typically includes system accounts, such as `root` and `daemon`, but you can add regular users to this file, as well. If your FTP server package doesn't install this file, check the server's documentation to see whether it's supported. If it's not, check to see whether the server provides another means of restricting who may access the system.

Anonymous access Most FTP servers provide some means to support anonymous access, but the details of how this is done vary from one FTP server to another.

The issue of anonymous access is an important one, so I provide a few examples. First, the `ftpd` server, as noted earlier, is designed to provide *only* anonymous access. It's therefore a good choice if you want to provide anonymous access only, and not regular user access.

If you use `ProFTPD`, the anonymous configuration requires setting quite a few options in `/etc/proftpd/proftpd.conf`:

```
<Anonymous /home/ftp>
User anonymous
Group anonymous
AnonRequirePassword off
<Limit LOGIN>
Allow from all
</Limit>
<Limit LIST NLST RETR MTDM PWD XPWD SIZE STAT CWD XCWD CDUP XCUP >
AllowAll
</Limit>
<Limit STOR STOU APPE RNFR RNT0 DELE MKD XMKD SITE_MKDIR RMD XRMD
SITE_RMDIR SITE SITE_CHMOD SITE_CHGRP >
DenyAll
</Limit>
</Anonymous>
```

You should change the directory in the first line (`/home/ftp` in this example) to point to the directory that holds the files you want to be available. A few other options are fairly obvious, but some are obscure. The second and third `<Limit>` blocks, in particular, specify what types of operations are permitted, such as `LIST` and `RETR` (to list and retrieve files, respectively). These codes can be confusing to the uninitiated. It's no surprise that the `GProFTPD` configuration tool exists, since it helps you set up anonymous (and nonanonymous) access using a point-and-click interface—but even `GProFTPD` can be perplexing.

The vsftpd server is simpler to configure than ProFTPd. To enable anonymous access, you need only ensure that one line is present in the `/etc/vsftpd/vsftpd.conf` file:

```
anonymous_enable=YES
```

You must also ensure that your system has an account for the `ftp` user. That user's home directory will be the directory used for anonymous FTP access.

Using FTP Clients

FTP clients are many and varied. One of the most common in Linux is the `ftp` program, which you can use by typing the program name followed by the name of the site you want to access, as in `ftp ftp.example.org`. The program then prompts you for a username and password. The result is an `ftp>` prompt, at which you type various commands, the most common of which are summarized in Table 11.3.

TABLE 11.3 Common ftp Commands

| Command Name | Effect |
|------------------------------|--|
| <code>ascii</code> | Set an ASCII (plain-text) transfer. |
| <code>binary</code> | Set a binary transfer; synonymous with <code>image</code> . |
| <code>bye</code> | Exit from the program; synonymous with <code>exit</code> and <code>quit</code> . |
| <code>cd dir</code> | Change to a new directory on the server. |
| <code>chmod mode file</code> | Change the mode (permissions) on the specified file. |
| <code>close</code> | Terminate an FTP session without exiting from the program; synonymous with <code>disconnect</code> . |
| <code>delete file</code> | Delete a file. |
| <code>dir</code> | Obtain a directory listing; synonymous with <code>ls</code> . |
| <code>disconnect</code> | Terminate an FTP session without exiting from the program; synonymous with <code>close</code> . |
| <code>exit</code> | Exit from the program; synonymous with <code>bye</code> and <code>quit</code> . |
| <code>get file</code> | Retrieve a file; synonymous with <code>receive</code> . |
| <code>help command</code> | Display help on the specified command (or a list of all commands); synonymous with <code>?</code> . |

TABLE 11.3 Common ftp Commands (*continued*)

| Command Name | Effect |
|---|--|
| image | Set a binary transfer type; synonymous with binary. |
| lcd <i>dir</i> | Change the current directory on the client system. |
| ls | Obtain a directory listing; synonymous with dir. |
| mdelete <i>files</i> | Delete multiple files. |
| mget <i>files</i> | Retrieve multiple files. |
| mkdir <i>dir</i> | Create a directory on the remote system. |
| mput <i>files</i> | Send multiple files. |
| open <i>server</i> | Open a connection to the specified remote system. |
| prompt | Toggle prompting to verify each file in multifile commands (mget, mput, etc.). |
| passive | Toggle passive transfer mode on and off. |
| put <i>file</i> | Send one file; synonymous with send. |
| pwd | Display the current directory on the server. |
| quit | Exit from the program; synonymous with bye and exit. |
| recv <i>file</i> | Retrieve a file; synonymous with get. |
| reget <i>file</i> | Retrieve a file starting with the end of the local file of the same name; useful for restarting an aborted transfer. |
| rstatus | Display status of server. |
| rename <i>oldname</i>
<i>newname</i> | Rename a file on the server. |
| rmdir <i>dir</i> | Remove a directory on the server. |
| send <i>file</i> | Send one file; synonymous with put. |
| status | Display current status. |



Type **help** or **?** to obtain a list of ftp commands. Type **help *command*** or **? *command*** to obtain a very brief description of what a specific *command* does.

Many other FTP clients are available, including GUI tools such as gFTP (<http://gftp.seul.org>). Web browsers can function as FTP clients; you enter the FTP server's hostname, preceded by `ftp://` for anonymous access, as in `ftp://ftp.example.org`. For nonanonymous access, you precede the hostname by the username and an at-symbol (@), as in `ftp://lucy@ftp.example.org`. The Web browser will then prompt you for a password.

One unusual but useful FTP client is fuseftp (<http://freshmeat.net/projects/fuseftp/>). In conjunction with the Filesystem in Userspace (FUSE; <http://fuse.sourceforge.net>) software, fuseftp enables you to mount an FTP server's files as if they were on an SMB/CIFS share or NFS export. This type of access is very handy if you want to directly access files on the FTP server without storing them locally, as in viewing graphics files or editing text files.

FTP is an old and quirky protocol, and it presents a couple of pitfalls that are unique to FTP. These are largely client-side problems, so users should be aware of them; however, if you configure a firewall, as described in Chapter 9, "Configuring Advanced Networking," you may need to take the first of these issues into account.

The first FTP quirk is in the way it manages ports. Two ports (TCP ports 20 and 21) are registered to FTP. Port 20 is the data port, which is used for data transfers; port 21 is the command port, which is used for issuing commands. The big problem with this configuration is that in the normal *FTP active mode*, the client initiates the connection to the command port, and then the server initiates a reverse connection to the client from the server's command port. This server-to-client connection is blocked by some firewalls, so FTP often fails on firewalls that aren't explicitly configured to permit this connection. A client-side workaround to this problem is to use *FTP passive mode* (by typing **passive** in the ftp program or by selecting equivalent options in GUI or other FTP clients). In passive mode, the FTP client initiates both connections, which overcomes the problem of firewalls that block all incoming connections. Passive mode uses an unprivileged port on the server for its data transfers, though, which can sometimes run afoul of firewalls. Thus, you may need to try both active and passive modes to determine which one works best.

The second FTP quirk relates to transfer mode. FTP clients often default to ASCII (plain-text) transfer mode, in which the files are transferred in such a way as to dynamically alter the character encoding to account for the different ways that different OSs store plain-text files. (Linux/Unix, Windows, and the original Mac OS all used different line-ending conventions, and some even more exotic systems use non-ASCII encoding methods.) Using ASCII mode works well for plain-text files, but this mode will almost always corrupt binary files, such as tarballs, graphics files, word processing documents, and so on. You can use the `ascii` and `binary` options in ftp, or similar options in other FTP clients, to set the transfer mode. If in doubt, use binary mode; most text editors today can handle any of the common line-ending types, so retrieving a text file in the wrong format will cause minimal or no problems. Some Linux configuration files must use Unix-style line endings, though, so you may need to pay attention to this detail if you transfer such files.

Configuring Web Servers

Web servers are another staple of the Internet. These servers handle the *Hypertext Transfer Protocol (HTTP)*, which is why most Web page addresses begin with the string `http://`. (A secure HTTP variant also exists; such pages are denoted by a leading `https://`.) Apache (<http://httpd.apache.org>) is the most popular Web server for Linux. Some distributions install Apache by default, but many don't, so if you want to run Apache, you may need to install it; it usually comes in a package called `apache` or `httpd`.



NOTE

Other Web server packages also exist, but none is nearly as popular as Apache. Because Apache usually ships with Linux and is sometimes installed by default, it's often a good choice, even though it provides more features than many sites need.

Apache is an extremely complex server; this section presents only the barest features of the server. To learn more, consult its documentation or a book on the subject, such as Charles Aulds' *Linux Apache Server Administration, 2nd Edition* (Sybex, 2002).

Setting Basic Apache Options

Once it's installed, Apache relies on a configuration file, which is likely to be called `httpd.conf` or `httpd2.conf` (the latter name most often applies to Apache 2.0 or later installations). This file usually appears in `/etc/apache`, `/etc/httpd`, or `/etc/httpd/conf`. In any event, the usual Apache configuration file consists of comment lines that begin with hash marks (`#`) and option lines that take this form:

Directive Value

Directive is the name of an option you want to set, and *Value* is the value you want to assign to *Directive*. This file also contains blocks of options, which are denoted by codes in angle brackets:

```
<IfDefine APACHEPROXIED>
  Listen 8080
</IfDefine>
```

A default Apache configuration typically delivers Web pages from a central location, which is specified with the `DocumentRoot` directive. Chances are you don't want to serve your distribution's generic Web page, so you should look for this directive and either change it to point to your own home page or replace the files in the default location with those you've created.

In addition to the main site Web page, Apache can deliver user Web pages, which it reads from a directory specified with the `UserDir` directive. These pages normally reside in a subdirectory of each user's home directory. For instance, if `UserDir` points to `public_html`, the `public_html` subdirectory of each user's home directory holds that user's Web pages, which can then be accessed by appending a tilde (`~`) and the username in the Web address, as in `http://www.asmallisp.net/~john/` to access john's home page.

The `ServerRoot` directive identifies the root location for the server itself. Other configuration and log files are identified relative to this directory.

Several Apache configuration options relate to logging. In particular, `ErrorLog` sets the filename of the error log file, `CustomLog` sets the access log file, `LogLevel` sets the system logger level, and `LogFormat` controls the format of data stored in the access log file. (Chapter 4 describes the system logger.)

Users or Web site maintainers can override some Apache configuration options using the `.htaccess` files in the directories that Apache serves. The format of the `.htaccess` file is just like that of the main Apache configuration file, but the options set in this file affect only the directory tree in which the `.htaccess` file resides. Normally, this file is used only by users whose personal Web pages are shared via a global `UserDir` directive or by Web site maintainers who may edit one or more subdirectories of the server's main Web space directory but who don't have full administrative access to edit the main Apache configuration file.

Using Apache Modules

Many Apache features are implemented in code that's optional. This code is sometimes compiled directly into the main Apache binary, but other times it's stored in separate modules. These modules are similar to Linux kernel modules: they're separate files that may be loaded into memory or ignored, depending on the needs and configuration of the system as a whole. To load a module, you use the `LoadModule` directive in the `httpd.conf` configuration file:

```
LoadModule module_name module_filename
```

The *module_name* is typically related to the *module_filename*, except that the latter includes the path relative to the server's root directory (set via `ServerRoot`) and is typically followed by `.so`. The two may also differ slightly in form. As an example, you might have a line like the following in your configuration file:

```
LoadModule auth_basic_module modules/mod_auth_basic.so
```

This line loads a module that provides some authentication features. Chances are your distribution's default Apache configuration includes quite a few `LoadModule` directives. You can learn what they do at <http://httpd.apache.org/docs/2.2/mod/>. (Change 2.2 to your Apache version, if it's not a 2.2.x version.)

Some modules may be compiled directly into your version of Apache. To see what modules are so compiled on your system, type **httpd -l** or **apache -l**, depending on the name of your Apache binary.

Configuring Scripts

Many sites run a Web server merely to deliver static content—that is, pages whose content doesn't change. Web servers can also run dynamic content, though, such as Common Gateway Interface (CGI) scripts, PHP: Hypertext Preprocessor (PHP; a recursive acronym, formerly expanded as *Personal Home Page*) scripts, or Java servlets. These scripts can extend the functionality of a Web server, enabling it to provide dynamic content or perform computing functions on behalf of clients. Each of these technologies is extremely complex, and

this section provides only enough information for you to activate support for it in Apache. If you need to maintain a site that relies on scripting technology, you should consult additional documentation on the topic.



Enabling scripting features on a Web server can be risky, because an incorrect configuration with buggy scripts can give an attacker a way to compromise the computer's security as a whole. Thus, I strongly recommend that you not attempt this unless you learn far more about Web servers and their scripting capabilities than I can present in this brief introduction to this topic.

CGI scripts are scripts or programs that run on the Web server at the request of a client. CGI scripts can be written in any language—C, C++, Perl, Bash, Python, or others. CGI scripts may be actual scripts or compiled programs, but because they're usually true scripts, the term *CGI script* applies to any sort of CGI program, even if it's compiled. The script must be written in such a way that it generates a valid Web page for users, but that topic is far too complex to cover here.

To activate CGI script support in Apache, you typically point to a special CGI directory using the `ScriptAlias` directive:

```
ScriptAlias /cgi-bin /usr/www/cgi-bin
```

This line tells Apache to look in `/usr/www/cgi-bin` for scripts. This directory may be a subdirectory of the parent of the `DocumentRoot` directory, but their locations can be quite different if you prefer.

PHP, by contrast, is a scripting language that's designed explicitly for building Web pages. As with CGI scripts, writing PHP scripts is a complex topic that's not covered on the Linux+ exam. You should, however, know how to activate PHP support in Apache. To begin this task, ensure that you've installed the necessary PHP packages. Chances are you'll need one called `php`, and perhaps various support or ancillary packages, too.

With PHP installed, you can configure Apache to support it. This is done via Apache configuration lines like the following:

```
# Use for PHP 5.x:
LoadModule php5_module      modules/libphp5.so
AddHandler php5-script php

# Add index.php to your DirectoryIndex line:
DirectoryIndex index.html index.php
AddType text/html          php
```



The preceding configuration works for PHP version 5. If you're using another PHP version, you may need to change the filenames.

The first couple of lines in this configuration simply load the PHP module and handler. The `DirectoryIndex` and `AddType` lines help Apache manage the PHP files. The `DirectoryIndex` line will replace existing lines in your configuration—or more precisely, you should ensure that `index.php` appears on the `DirectoryIndex` line along with any other filenames you use for index files.

In addition to these global options, directories that hold PHP scripts may include files called `php.ini`, which set various PHP interpreter options. There are quite a few options, such as `user_dir`, `include_path`, and `extension`. If you need to tweak your PHP settings, I recommend starting from a sample file, such as the global `php.ini` file in `/etc`.

The final Web server scripting solution is Tomcat (<http://tomcat.apache.org>), which enables Apache to run Java servlets on the server computer. Tomcat is conceptually similar to PHP. As with PHP, you must begin by installing appropriate Tomcat packages on your system. The main Tomcat package is likely to be called `tomcat5`, `tomcat6`, or something similar. (Version 6 is the latest version of Tomcat as I write, but version 5 is still available in many distributions.) High-level package tools, such as Yum or APT, will help you locate the appropriate packages.

With Tomcat installed, you must tell Apache how to use it. Typically, Tomcat packages install their own configuration files with options to enable support in Apache. To use them, you need add only one line to your Apache configuration file, such as this one:

```
include /tomcat/conf/mod_jk.conf-auto
```

The precise file you reference may vary from one distribution to another, so you should check your own Tomcat package to see what configuration files it's installed.

Whatever scripting tool you use, you can restart Apache via its SysV startup script to have it enable scripting support. It's then up to you or your Web developers to create appropriate scripts to manage dynamic content on your site. This is a very complex topic that's not covered on the Linux+ exam.

Configuring Virtual Hosting

Another Web server feature that's handy on large systems is *virtual hosting*—one server that hosts multiple Web sites. Suppose two organizations with two domains (say, `example.com` and `pangaea.edu`) both want to host Web sites, but to reduce costs, they decide to share a single computer to do the job. Both point hostnames in their domains to this computer's IP address. Virtual hosting enables the computer with this IP address to respond differently depending on the hostname the user enters in a remote Web browser. Web hosting ISPs make heavy use of this feature, supporting many domains on a single computer. It can also be handy if you've changed your company name—you can run a single server that responds to both old and new domain names, with a notice about the change on the old name.

Implementing virtual hosting can be done in a couple of ways. One is to create a block with the `VirtualHost` directive:

```
<VirtualHost *>
  ServerName www.example.com
```



```
DocumentRoot /usr/www/example/html
</VirtualHost>
```

Directives inside this block apply only when the client contacts the server using the hostname specified on the `ServerName` line. A second method involves the `VirtualDocumentRoot` directive, which specifies a document root directory that incorporates the hostname. This is specified with a special code that takes the form `%N.M`, where *N* is the hostname component and *M* is the number of characters (all characters, if it's omitted). A negative number counts from the final component. For instance, with a hostname of `www.sales.example.com`, `%-2` expands to `example` and `%4.2` expands to `co`. This code is incorporated into a directory specification:

```
VirtualDocumentRoot /usr/www/%-2.1/%-2
```

Controlling Apache

You can start, stop, and restart Apache via its SysV startup script, just as you can control many other services. (Chapter 4 describes SysV startup scripts in detail.) Another tool, `apachectl`, provides similar capabilities, plus some more. In fact, Apache SysV startup scripts often work by invoking `apachectl`.

Typically, you'll call `apachectl` by typing the utility's name along with an option, the most common of which are summarized in Table 11.4.

TABLE 11.4 Common `apachectl` Commands

Command Name	Effect
<code>start</code>	Launches Apache.
<code>stop</code>	Terminates Apache.
<code>graceful-stop</code>	Similar to <code>stop</code> , but requests that are currently being serviced are permitted to complete.
<code>restart</code>	Restarts Apache. If it's not running, <code>restart</code> is identical to <code>start</code> .
<code>graceful</code>	Similar to <code>restart</code> , but requests that are currently being serviced are permitted to complete.
<code>fullstatus</code>	Displays a status report, including a list of requests being serviced. This option requires the <code>mod_status</code> module enabled.
<code>status</code>	Similar to <code>fullstatus</code> , but omits the list of requests being serviced.
<code>configtest</code>	Performs a test of the configuration file syntax and reports any errors.

You might use `apachectl` rather than the Apache SysV startup script if you need to get a status report or check the syntax of your configuration file. You may want to check your SysV startup script to see whether it uses the normal or graceful options for stopping and restarting.

Using Web Proxy Servers

A *proxy server* is a program that accepts network access requests on behalf of a client, accesses the target server, and relays the results back to the client. In some respects, a proxy server is similar to a firewall computer; however, a proxy processes access requests at a higher level. For instance, a Web proxy server parses the URLs sent by clients and can fully assemble the Web pages sent in response. This enables a proxy server to use high-level data to block undesirable Web pages, to cache data for quicker subsequent accesses, or to perform other high-level tasks.

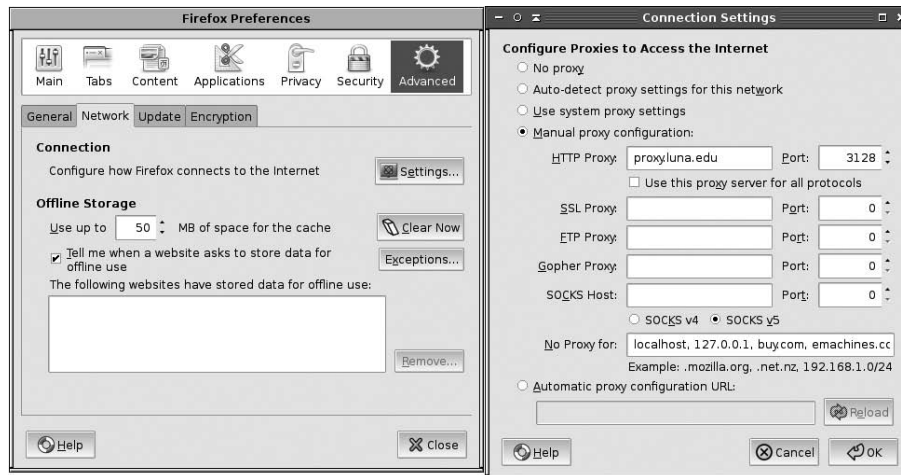
Although proxy servers can exist for many protocols, they're very common for HTTP, the Web protocol. Proxy servers can be used to block sites that might not be suitable for children; to limit employees' nonwork use of the Web from employer computers; to check Web pages for malware; or to cache Web pages locally, thus improving performance, to name just four common uses of Web proxies.

Squid (<http://www.squid-cache.org>) is a popular Web proxy server for Linux. Its emphasis is on caching data for speed, rather than providing security or other features. If you install Squid and then immediately launch it, the program will work in this capacity immediately; however, its configuration file, `/etc/squid/squid.conf`, provides a dizzying array of options. If you need to adjust Squid's configuration, you can peruse this configuration file, preferably in conjunction with Squid's documentation.

You can install Squid on an individual client computer to gain some benefits; however, Squid works best when it runs on a central server computer and caches requests from multiple clients. For instance, if Fred accesses a Web page from his desktop system, a Squid proxy running on a different but nearby system can cache that page locally. If Mary then accesses the same page from her computer, Squid can deliver the cached page. If Squid were installed separately on Fred's and Mary's computers, it wouldn't be able to deliver the cached copy to Mary, thus eliminating Squid's benefits.

Squid is a proxy server, and as a server it must be run in one of the ways appropriate to servers, as described in Chapter 4. Typically, this means that Squid is run from a SysV startup script. If you've just installed Squid, you should be sure to launch it manually the first time and ensure that it's configured to run automatically when you reboot the computer.

In addition to installing and running Squid on its host system, you must configure clients to use it. This is typically done by selecting appropriate options in your clients' Web servers. For instance, in Mozilla Firefox on Linux, you should select **Edit > Preferences** to reveal the Firefox Preferences dialog box. Select the **Advanced** option, click the **Network** tab, and click the **Settings** button. The result will be the **Connection Settings** dialog box, shown in Figure 11.1 along with the Firefox Preferences dialog box. Select **Manual Proxy Configuration**, and enter the hostname or IP address of the proxy server computer, along with the port number it's using. (Squid defaults to port 3128.)

FIGURE 11.1 You must configure Web browsers to use a proxy server such as Squid.

Alternatively, you can use an iptables firewall configuration on your network's router to redirect all outgoing Web traffic to the proxy server. (Chapter 9 describes iptables.) This configuration obviates the need to configure each client individually; however, it also means that if the proxy server corrupts data, your users will have no recourse. If you attempt such a configuration, be sure to exempt the proxy server computer from the rule, lest you set up an infinite loop in which the proxy server's traffic is redirected to itself!

Using Text-Based Web Clients

You're probably already familiar with the most common Web browsers: GUI tools such as Mozilla Firefox, Konqueror, and Opera. A few text-based Web clients deserve mention, though: lynx, wget, and curl.

The lynx program is a text-based interactive Web browser. It has features that are similar to those of GUI Web browsers, but it doesn't support graphics. You browse the Web with lynx by using your keyboard's arrow keys to move a cursor from one link to another and then pressing the Enter key to select the link. lynx is particularly useful if you need to browse the Web using a text-only display. Users with visual impairments also use lynx in conjunction with screen reader software.

The wget program enables you to retrieve a document from a Web server and store it locally—think of it as analogous to Linux's cp command, but using a source document that's a Web URL. For instance, suppose you want to retrieve the main page from the <http://www.linux.org> Web site. You could do so with the following command:

```
$ wget http://www.linux.org
```

In this case, `wget` will locate the main document (`index.html` in this case, although the filename could be different), retrieve it, and save it to disk. If you want to retrieve something other than the main page, you can specify its filename:

```
$ wget http://www.linux.org/apps/all/Administration/Backup.html
```

You can use quite a few options with `wget`; consult its man page for details. One option in particular deserves mention: `-r`, which causes `wget` to retrieve a site recursively—it copies the document you specify and all those documents from the same site that are referenced by it. This can be a handy way to back up the contents of a Web server from another computer, either because you maintain the site and need a local copy or because you want to retrieve a site for later offline reading.

The final text-mode Web command is `curl`. This tool is similar to `wget` in that it's designed to transfer files to and from servers. By default, though, `curl` displays the retrieved file on standard output, similar to `cat` for local files. This makes it handy for directly viewing text files retrieved from servers, but it's less useful than `wget` for backing up an entire site.

Although I've described `wget` and `curl` as Web retrieval tools, both support additional protocols. `wget` works with HTTPS (secure HTTP) and FTP, while `curl` works with these protocols along with TFTP, Telnet, and several others.

Using Windows Remote Access Tools

Chapter 9 presented information on remote access using some common Linux tools, including SSH and the X Window System. These aren't the only remote access protocols available, however. A couple of tools are particularly useful for interoperating with Windows systems. The first of these is `rdesktop`, which uses the Remote Desktop Protocol (RDP) to connect to Windows systems that run Terminal Services to enable remote GUI logins. The second tool is Virtual Network Computing (VNC), which is a cross-platform GUI login tool that uses the Remote Framebuffer (RFB) protocol. You can configure a VNC server in Linux to enable logins from other systems, and you can use a VNC client to log in to remote systems.

Using *rdesktop*

The `rdesktop` program is fairly straightforward to use, as you can see by its syntax:

```
rdesktop [options] server[:port]
```

Table 11.5 summarizes the most common `rdesktop` options. Consult the program's man page for a more complete list. Most of the options not shown in Table 11.5 are highly technical and relate to the way the protocol transfers data or to enabling the server to access local hardware devices on the client.

TABLE 11.5 Common rdesktop Options


Command Name	Effect
-u <i>username</i>	Specify a username for accessing the server.
-d <i>domain</i>	Specify a domain for accessing the server.
-s <i>shell</i>	Start the specified program rather than Explorer on the server.
-p <i>password</i>	Pass the specified password to the server. This may have no effect if the Always Prompt for Password option is set on the server.
-g <i>geometry</i>	Specify the desktop size; <i>geometry</i> is specified as width and height, separated by an x, as in 1024x768.
-f	Enter full-screen mode.
-a <i>bitdepth</i>	Specify the color depth (8, 15, 16, or 24). Depths greater than 8 work only with Windows XP or newer.

Using VNC

VNC’s greatest strength is its cross-platform nature. VNC clients and servers are available for Linux, other Unix variants, Mac OS, Windows, and more obscure OSs. This makes VNC a good choice for remote GUI access in a cross-platform environment.

Configuring a VNC Server

The standard VNC server configuration on Linux is rather odd: it runs the VNC server as an ordinary user. Accessing the server produces a login to the computer as the user who ran the server. Thus, if Fred and Mary both run VNC servers on a Linux system, each can log in separately; but if George and Sally also have accounts on the system but don’t run VNC, they won’t be able to log in remotely. Behind the scenes, VNC server programs for Linux include both a VNC server and an X server; the two are linked, with the VNC server half functioning as a virtual keyboard, mouse, and monitor for the X server half.

**NOTE**

It’s possible to link VNC to an X Display Manager Control Protocol (XDMCP) server. When this is done, users who access the VNC server system will see a standard Linux XDMCP login screen. This configuration requires using advanced VNC options that aren’t described here.

The first step to running VNC is to install it. Several different VNC servers exist, and most distributions ship with at least one of them. Typically, separate client and server packages are available, often called `vnc` and `vnc-server`, or variants of this (such as `tightvnc` and `tightvncserver`). If you're not sure whether VNC is installed on your system, use your high-level package manager, such as `yum` or `apt-get`, to search for relevant packages. If VNC isn't installed, install the appropriate packages for your distribution, or check the RealVNC Web site (<http://www.realvnc.com>) or the Tight VNC Web site (<http://www.tightvnc.com>). After you install VNC, follow these steps *as an ordinary user* on the VNC server system:

1. Create a directory called `.vnc` in your home directory. This directory will house your VNC configuration files.
2. Type **`vncpasswd`**. The program prompts you for a password and for a verification of this password. VNC doesn't, by default, use the normal Linux password database. You will need this password to gain entry to the system.
3. Type **`vncserver`**. This command is actually a script that performs various checks and then starts a VNC server (using the `Xvnc` program file) in your name. The program displays some summary information, including a VNC session number:

```
New 'tranquility.luna.edu:1 (neil)' desktop is tranquility.luna.edu:1
```

You should now be able to access the VNC server system, as described in the upcoming section “Using a VNC Client.” When you try this, though, you may run into problems. One common issue is that the server fails to start, or it starts and then crashes. Check for log files in `~/ .vnc`. These files may include clues to the problem. One common problem relates to font paths; the VNC server is very sensitive about its font paths, and it will crash if you specify font directories that don't exist or that are misconfigured. You can add the `-fp` option and a comma-separated list of font directories to the command line to work around this problem. Alternatively, you can adjust your configuration files.

A second problem—or series of problems, really—is that various default options may be set strangely. Most configurations use two types of configuration files:

The VNC Startup Script The `vncserver` script includes within it various options, such as the default desktop size, the default font path, and so on. For the most part, these defaults are equivalent to X server defaults you would set in your local X server's `xorg.conf` file. You can edit this script to change these defaults.

User Configuration Files Most VNC servers use `~/ .vnc/xstartup` as a user's local startup script. This script may call `/etc/X11/xinit/xinitrc` or some other local default startup script, or it may launch a bare-bones window manager, such as `twm`. In any event, you can edit this script much as you'd edit any other X startup script.

Using a VNC Client

The Linux VNC client program is usually called `vncviewer`. This tool takes several options, but in most cases you use it by typing the program name followed by the VNC

server name, a colon, and the VNC session number. For instance, you might type the following command:

```
$ vncviewer blueox.luna.edu:1
```

```
VNC server supports protocol version 3.8 (viewer 3.3)
```

```
Password:
```

If you type the correct password, `vncviewer` displays more text in your terminal and opens a window displaying the remote computer's desktop. VNC clients are also available for many other OSs; check the original VNC or the Tight VNC Web sites for these clients. These non-Linux VNC clients work much like the Linux VNC client, but they emphasize dialog boxes for entering the server's hostname and password.

One important drawback of VNC is that the user must ordinarily launch the VNC server from the server computer before running the VNC client program from the client computer. In some cases, this requirement isn't a major issue. For instance, if you're sitting at a workstation and know you'll want to use it from another location in the near future, you can launch the server before leaving the workstation. In other cases, though, you may need to log into the VNC server computer using a text-mode login tool in order to launch the VNC server. This procedure requires two logins, which is a nuisance. What's more, this means you must run two login protocols on the server computer, increasing its security exposure.

Deploying MySQL

The Structured Query Language (SQL), as its expanded name suggests, is a language used for retrieving data from a database. In practice, SQL is implemented in several different database products. Thus, you should know a little about the SQL products that are available for Linux. With a SQL package installed, you can begin learning about the principles of SQL use and run a SQL product.

Picking a SQL Package

SQL is a language for accessing data, and specific SQL packages implement that language. This distinction is similar to that between a network protocol (such as SMTP) and the servers that implement it (such as sendmail and Postfix). In principle, you can use any SQL package to satisfy your SQL database needs. In practice, specific products that store data using SQL may work better with (or even require) particular packages. Some of the more common choices in Linux include the following:

MySQL Sun owns this SQL implementation, which it has released under the GPL. Most major Linux distributions include MySQL in their package databases. For a complete installation, you'll probably need to install multiple packages, such as a client, a server, and perhaps development tools. You can learn more at <http://www.sun.com/software/products/mysql/>.

PostgreSQL This SQL implementation evolved from the earlier Ingres software (the name *PostgreSQL* is a compressed form of *post-Ingres SQL*). It's available under the BSD license and is available as multiple packages in most Linux distributions. As with MySQL, you'll most likely have to install a client, a server, and perhaps additional support packages. PostgreSQL is headquartered at <http://www.postgresql.org>.

SQLite This package, based at <http://www.sqlite.org>, is a library that implements SQL. As such, it's not a stand-alone database; instead, it's intended as a way to provide programs with access to SQL database features within the program. If you install a program that uses SQLite, your distribution's package manager should install the relevant libraries for you. If you want to write a program that requires database access and you don't want to install a complete client-server SQL package such as MySQL or PostgreSQL, SQLite may be just what you need.

There are dozens more SQL database products for Linux. For the purpose of learning SQL, MySQL or PostgreSQL should do fine, or you can use another full implementation if you prefer. If you have a specific purpose in mind for using SQL, though, you should research SQL packages in more detail. You may need a particular product for compatibility with other software, or you may need a SQL package that provides specific features. The Linux+ objectives mention only MySQL by name.

As just noted, some SQL packages, including MySQL and PostgreSQL, operate on a client-server model: one program (the server) manages the database, while another (the client) provides users and programs with access to the database. Such implementations can work over a network, enabling users at multiple client systems to access a centralized database server.

Using MySQL

To learn about SQL, you should have access to a SQL database. For purposes of demonstration, I'm using MySQL as a reference. Other SQL implementations are similar to what I describe here, but some details differ. From an administrative perspective, you need to know how to set MySQL configuration options, how to start and stop the server, and how to test that it's working. Actually performing database queries with MySQL is a complex topic that's beyond the scope of this book or the Linux+ exam.

Configuring, Starting, and Stopping MySQL

MySQL's main configuration file is called `my.cnf`, and it's usually stored in `/etc` or `/etc/mysql`. You should peruse this file to familiarize yourself with some of the key MySQL options. Unless you know you need to change the defaults, though, you should be cautious about changing this file's contents, at least initially.

One initial configuration task you must perform is to set the MySQL root password. Just like Linux itself, MySQL requires an administrative (root) user; however, MySQL's root isn't the same as the Linux root account. A default MySQL installation is likely to

require no password for administrative access. To enable a password, type the following command:

```
$ mysqladmin -u root password newpassword
```

Change *newpassword* to the password you want to use, of course. If your system already has a root password set, you can change it with another command:

```
$ mysqladmin -u root -p'oldpassword' password newpassword
```



Some distributions, such as Ubuntu, include installation scripts that prompt you to enter a MySQL root password when you install the software. Thus, you might not need to type these commands.

Typically, MySQL is started and stopped using SysV startup scripts, such as `/etc/init.d/mysql`. Chapter 4 describes SysV startup scripts in more detail, so consult it if you need help starting or stopping MySQL. You can verify that MySQL is running using `ps`; search for a process called `mysqld`:

```
# ps ax | grep mysql
1948 ?      S        0:00 /bin/sh /usr/bin/mysqld_safe
2013 ?      Sl       33:30 /usr/sbin/mysqld --basedir=/usr
--datadir=/var/lib/mysql --user=mysql --pid-file=/var/run/mysqld/mysqld.pid
--skip-locking --port=3306 --socket=/var/run/mysqld/mysqld.sock
2014 ?      S        0:00 logger -p daemon.err -t mysqld_safe -i -t mysqld
30933 pts/0  S+       0:00 grep mysql
```

In this example, the `/usr/sbin/mysqld` program is the MySQL server, so it's definitely running. This output also shows the options used to launch the program. Three other MySQL-related processes were also found by this query, including the `grep` command used to search for MySQL. Be sure that you find `mysqld` (the daemon, whose name ends in `d`); the `mysql` program is the client, and it could be running even if the server isn't running. If you can't find the MySQL daemon running, try using its SysV startup script, and if that fails, search your log files for clues to what might be going wrong when you try to launch the server.

Testing the MySQL Connection

To test MySQL's basic operation, you should first ensure that the server is running, as just described. You can then start the SQL client. In the case of MySQL, this program is called `mysql`:

```
$ mysql
```

If the client can connect to a server, you'll see a `mysql>` prompt. If the server isn't running or can't be contacted, you'll see an error message. To verify the status of the `mysql` client connection to its server, you can use the `STATUS` command:

```
mysql> STATUS;
-----
mysql Ver 14.12 Distrib 5.0.51a, for debian-linux-gnu (x86_64) using readline 5.2

Connection id:          29
```

Several additional lines of status information will appear, as well. You can peruse these to learn a few more details, such as what version of the server is in use and how long it has been running.

If you've just installed MySQL, it may have no databases defined. To learn what's defined, you can use the `SHOW DATABASES` command:

```
mysql> SHOW DATABASES;
+-----+
| Database          |
+-----+
| information_schema |
+-----+
1 row in set (0.00 sec)
```

These operations should be more than sufficient to verify basic functioning of MySQL. You might not get in so easily, though, particularly if you use a computer other than the one on which the MySQL server is running to test the server's operation. (If network operation is important in your installation, you should definitely test network access to ensure that firewalls or other network issues won't cause problems.) If you receive a `Can't connect to MySQL server` message, chances are either the MySQL server isn't running or a network configuration is blocking access to the server. If you receive an `Access denied` message, the problem is with authentication—you must specify a legal account and password.

To connect to a remote server, use a particular MySQL account, or send a particular MySQL password, you must use the `--host` (or `-h`), `--user` (or `-u`), or `--password` (or `-p`) options, respectively. For instance, to connect to the MySQL server running on `third.luna.edu` as `dbuser` with a password of `e4bUg7TQ`, you might type the following:

```
$ mysql --host=third.luna.edu -u dbuser -pe4bUg7TQ
```

**NOTE**

When using the long option names, you must include an equal sign between the option name and its value, as in the `--host` option in this example. An equal sign is *not* used with the short option forms. For the password, you must *not* include a space between the short option name (`-p`) and the password. You might prefer to use the long form (as in `--password=e4bUg7TQ`) to improve legibility. If you include the `-p` option but omit the password, `mysql` will prompt for it.

Summary

Network servers are many and varied, and Linux provides programs to handle all common network protocols. These include DHCP for configuring clients' network settings, DNS for resolving hostnames and IP addresses, NTP for managing time on a network, SMB/CIFS for file sharing with Windows, NFS for file sharing with Linux and Unix, FTP for cross-platform file transfers, HTTP for Web browsing, VNC for cross-platform GUI access, SMTP for push e-mail, POP and IMAP for pull e-mail, and SQL for database access.

Although these servers share certain common features, such as their ability to be launched via SysV startup scripts or super servers, each protocol and server is unique. Indeed, for many protocols, multiple servers are available, and each server can have its own unique configuration style. Thus, you must study each server you install and run to determine how best to manage it.

Exam Essentials

Explain where Samba and NFS are best deployed. Samba is an implementation of the SMB/CIFS protocol suite, which is most often used for file and printer sharing by Windows systems; thus, Samba is best used as a server for Windows clients. NFS, by contrast, was designed as a file sharing protocol for Unix systems, so it's best used for sharing files with Unix or Linux clients.

Describe the Samba and NFS configuration files. Samba is configured via `smb.conf`, which is typically stored in `/etc`, `/etc/samba`, or `/etc/samba.d`. Global Samba options appear in the `[global]` sections, and subsequent sections define specific shares. NFS is configured via `/etc/exports`, which lists the directories that are to be exported, one per line. Each line also contains a list of computers or networks that may mount the export, along with mount options.

Summarize the differences between FTP and HTTP. The File Transfer Protocol (FTP) is a login-based protocol for transferring files. The Hypertext Transfer Protocol (HTTP) doesn't typically require logins (although some Web sites do implement login functionality). FTP is an old protocol with some quirks, such as requiring the use of two network connections. The World Wide Web (WWW or Web for short) is built atop HTTP, making HTTP an extremely important protocol on today's Internet.

Describe common Web server scripting tools. The Common Gateway Interface (CGI) is a general mechanism that enables a Web server to run a normal Linux program or script (written in Perl, Python, C, or any other language that Linux supports) to generate Web pages dynamically. PHP and Tomcat are more specialized languages that are designed with Web use in mind. They require additional configuration in Samba but can be easier to program for Web purposes.

Explain the purpose of rdesktop and VNC. The rdesktop program enables access to Windows computers that provide the Windows-specific Remote Desktop Protocol (RDP) for remote login access. VNC can fill a similar role, but VNC is a cross-platform tool; the server can run Windows, Mac OS, Linux, or various other OSs.

Describe the purpose of SQL and MySQL. The Structured Query Language (SQL) is a common database language; it's used to store data that can be retrieved by individuals or programs. SQL is network-enabled, meaning that database users can exist on computers other than the one that hosts the database. MySQL is a common implementation of SQL for Linux; it includes client, server, and support programs.

Review Questions

1. How does an NFS server determine who may access files it's exporting?
 - A. It uses the local file ownership and permission in conjunction with the client's user authentication and a list of trusted client computers.
 - B. It uses a password that's sent in unencrypted form across the network.
 - C. It uses a password that's sent in encrypted form across the network.
 - D. It uses the contents of individual users' `.rlogin` files to determine which client computers may access a share.
2. You want to export the `/home` directory to two computers via NFS: `remington` should have full read/write access, while `gentle` should have read-only access. How would you configure this in `/etc/exports`?
 - A. `remington(/home,rw) gentle(/home,ro)`
 - B. `[homes] remington(readwrite) gentle(readonly)`
 - C. `/home remington(rw) gentle(ro)`
 - D. `remington(/home,readwrite) gentle(/home,readonly)`
3. You want to temporarily export the `/mnt/cdrom` directory using NFS so that `reader.example.org` may read but not write the export. Assuming an NFS server is already running, what would you type at a shell prompt to accomplish this goal?
 - A. `exports -o ro reader.example.org:/mnt/cdrom`
 - B. `showmount -o ro reader.example.org:/mnt/cdrom`
 - C. `exportfs -o ro reader.example.org:/mnt/cdrom`
 - D. `mount -o ro reader.example.org:/mnt/cdrom`
4. A Samba server (`dance`) includes a `[homes]` share definition but no `[sammy]` share definition. Assuming the relevant account exists, what will then happen when the user `sammy` on a client attempts to access `\\dance\sammy`?
 - A. An error message will appear because the `[sammy]` share doesn't exist.
 - B. If the user enters the correct password, he'll be given access to the `/home` directory on the server.
 - C. The user will be given access to the `/tmp` directory whether or not a correct password is entered.
 - D. If the user enters the correct password, he'll be given access to his home directory's files on the server.

5. You're configuring a Samba server to participate in an existing Windows domain that's managed by a Windows Server 2003 domain controller. You want users to be able to authenticate using the Windows controller's account database. How would you set the `security` option in `smb.conf` to achieve this result? (Choose all that apply.)
- A. `security = User`
 - B. `security = Server`
 - C. `security = Domain`
 - D. `security = ADS`
6. What does the following line in an `smb.conf` file mean?
- ```
name resolve order = lmhosts
```
- A. Samba uses the `lmhosts` file preferentially for name resolution but will use other methods if necessary.
  - B. Samba uses the `lmhosts` file exclusively for name resolution and does not fall back on other methods.
  - C. Samba uses the `lmhosts` file as source material when it functions as a NetBIOS name server.
  - D. Samba uses the `lmhosts` file's contents to determine the priority given to different clients' name-resolution requests.
7. Which of the following is an important difference between RDP and VNC?
- A. RDP is a GUI protocol, but VNC is text-based.
  - B. RDP is a text-based protocol, but VNC is a GUI tool.
  - C. RDP is used primarily by Windows servers, but VNC is cross-platform.
  - D. RDP is used primarily by Unix and Linux servers, but VNC is used primarily by Windows servers.
8. A user wants to access a Linux desktop from a Windows computer. After configuring VNC, what would the user type at a Linux command prompt to enable this access?
- A. `vncserver`
  - B. `vncviewer`
  - C. `Xvnc`
  - D. `vncpasswd`
9. Which is the *best* way to launch an FTP server?
- A. From a SysV startup script.
  - B. From a super server.
  - C. From a local startup script.
  - D. It's impossible to say without more information.

10. You want to enable individual users to manage their personal Web pages on `www.example.net`. What Apache directive do you use to set the subdirectory within the users' home directories that will house their personal Web pages?
  - A. `ServerDir`
  - B. `ServerRoot`
  - C. `DocumentRoot`
  - D. `UserDir`
11. Which of the following tools caches Web (HTTP) accesses by clients, thus improving performance on subsequent accesses to the same popular sites?
  - A. Squid
  - B. PHP
  - C. lynx
  - D. CGI
12. You've been told that an Apache server is configured with Tomcat. What does this mean?
  - A. The server employs a GUI configuration tool for remote administration.
  - B. The server can run an enhanced version of `cat` locally to modify Web pages.
  - C. The server can run Java servlets locally to generate dynamic content.
  - D. The server supports Secure Sockets Layer (SSL) encryption.
13. Your employer has taken over the `example.org` domain, and you've been given the task of retrieving the current `example.org` Web site so that others may edit the files. This site is hosted at `www.example.org` and consists entirely of static Web pages. What might you type to retrieve these files?
  - A. `curl -r http://www.example.org`
  - B. `wget -r http://www.example.org`
  - C. `apache http://www.example.org`
  - D. `lynx http://www.example.org`
14. You launch the `ftp` client program to retrieve the latest Linux kernel from `ftp.kernel.org`. What `ftp` command will you use to retrieve the kernel file, once you've located it?
  - A. `cp`
  - B. `download`
  - C. `get`
  - D. `scp`

15. You've launched `ftp` to transfer some files, but you've just realized that you launched it in the wrong local directory; you want to be in `~/programming`. How can you correct this mistake without exiting `ftp`?
  - A. Press `Ctrl+Z`, type `cd ~/programming`, and then type `fg`.
  - B. Type `cd ~/programming`.
  - C. Type `change-to ~/programming`.
  - D. Type `lcd ~/programming`.
16. A Web server holds a text file for which you have a URL: `http://www.example.org/text.txt`. You want to view this text file without launching a full Web browser. How might you do this?
  - A. Type `ftp http://www.example.org/text.txt`.
  - B. Type `telnet http://www.example.org/text.txt`.
  - C. Type `wget http://www.example.org/text.txt`.
  - D. Type `curl http://www.example.org/text.txt`.
17. You've connected to a public FTP server and retrieved several JPEG graphics files. You can't open these files in any Linux graphics editor, though. What is the most likely explanation?
  - A. The FTP transfer occurred in ASCII mode, thus corrupting the binary graphics data.
  - B. The FTP transfer occurred in passive mode, thus corrupting the binary graphics data.
  - C. FTP is unsuitable for transferring graphics data; you should have used HTTP.
  - D. The graphics files were encrypted, so you need a private key to decrypt them.
18. When you attempt to download a file from an FTP site using a computer located on your heavily protected local network, nothing seems to happen. The server responds to pings, but you can't download files. What might you do to overcome this problem?
  - A. Type `ascii` at the `ftp>` prompt, and try again.
  - B. Type `passive` at the `ftp>` prompt, and try again.
  - C. Type `mput fix*` at the `ftp>` prompt, and try again.
  - D. Type `mget fix*` at the `ftp>` prompt, and try again.
19. You want to set the password on a freshly installed MySQL server. What command would you use to do this?
  - A. `mysqladmin -u root password newpassword`
  - B. `mysqlpasswd newpassword`
  - C. `passwd mysql`
  - D. `smbpasswd mysql`
20. Which option would you pass to `mysql` to have it access a remote MySQL server?
  - A. `--server=name`
  - B. `--host=name`
  - C. `--remote=name`
  - D. `--db=name`



# Answers to Review Questions

1. A. NFS uses a “trusted host” policy to let clients police their own users, including access to the NFS server’s files. NFS does not use a password, nor does it use the `.rlogin` file in users’ home directories.
2. C. Option C presents the correct syntax for achieving the specified goal in `/etc/exports`. Options A and D incorrectly place the exported directory name in the option list for each client. Option B uses `[homes]` (a Samba name for users’ home directories) rather than `/homes`. Options B and D incorrectly expand the `ro` and `rw` codes into `readonly` and `readwrite`.
3. C. The `exportfs` program controls the NFS server; it adds or removes directories and clients from the list the server maintains, thus temporarily extending or restricting the list that’s normally maintained in `/etc/exports`. Option C presents the correct syntax for this program to achieve the stated goal. There is no standard `exports` command, so option A is incorrect. Option B’s `showmount` command displays information on the clients that are using the server, but it doesn’t change the export list. The `mount` command mounts a remote export; it doesn’t affect what’s exported, so option D is incorrect.
4. D. The `[homes]` share in Samba is special; it gives access to users’ home directories, with each user being given access to his or her own home directory, as option D describes. Option A is incorrect because the point of the `[homes]` share is to enable access to home directories without having to explicitly define a new share for each user. Option B is incorrect because the `[homes]` share gives access to users’ individual home directories, not to the Linux `/home` directory, which is typically the directory in which all users’ home directories reside. Option C is incorrect because a correct password is still normally required to access `[homes]` and because this share doesn’t give access to `/tmp` unless options are set strangely. (The default directory for most Samba shares is `/tmp`, but this isn’t true of `[homes]`.)
5. B, C, D. The `Server` setting tells Samba to authenticate against the domain controller without fully joining the domain. The `Domain` setting tells Samba to fully join the domain using Windows NT 4 protocols. The `ADS` setting tells Samba to fully join the domain using Active Directory (AD) protocols. Any of these options will work, if properly configured. The `User` setting tells Samba to use its local account database, so this setting won’t do as the question specifies.
6. B. The `name resolve order` option in Samba determines what tools Samba uses to resolve hostnames into IP addresses. Since only one option (`lmhosts`) appears in this example, this is the only tool that’s used, as stated by option B. Option A is incorrect because no other methods will be used. Options C and D are incorrect because the `name resolve order` option controls Samba’s own name resolution, not how it functions as a NetBIOS name server or how it delivers names to clients.
7. C. The Remote Desktop Protocol (RDP) is used by Windows systems to support remote GUI logins. (The Linux `rdesktop` program is an RDP client.) The Virtual Network Computing (VNC) software is a cross-platform remote GUI login tool using the Remote Framebuffer (RFB) protocol. Thus, option C is correct. Since both tools are GUIs, options A and B are both incorrect. Option D would more accurately describe X (used by Unix and Linux) vs. RDP (used by Windows).

8. A. The `vncserver` script starts the VNC server as an ordinary user, enabling subsequent access to the Linux computer from remote systems. The `vncviewer` program is the VNC *client*; the user might use this program on a remote Linux system to access the server that's launched by `vncserver`. The `Xvnc` program is the actual VNC server binary, but it must be launched with various options—that's what the `vncserver` script does. The `vncpasswd` utility sets a VNC password; the command specifies that VNC has already been configured, so this step isn't necessary at this point.
9. D. FTP server programs vary in design; some are intended to be run from a local or SysV startup script, others are intended to be run from super servers, and others can be launched in either way. The needs of the site must also be considered; for instance, a lightly used FTP server might best be launched from a super server, whereas an FTP server that's in constant use might better be launched from a startup script. Thus, option D is correct.
10. D. The `UserDir` directive sets the subdirectory name that will be served as personal Web pages. The `ServerDir` directive is fictitious. `ServerRoot` sets the base directory where Apache looks for various files; many other options are specified relative to this one. The `DocumentRoot` directive sets the location of the server's *main* home page, rather than individual users' *personal* home pages.
11. A. The Squid program is a caching proxy server, meaning that it provides the features described in the question. PHP is a tool for running Web-centric scripts. `lynx` is a text-based Web browser. CGI is the Common Gateway Interface, a tool for running scripts from a Web server.
12. C. Tomcat refers to a package that enables an Apache server to run Java servlets, as described in option C. Tomcat has nothing to do with remote GUI configuration, the `cat` utility, or SSL encryption.
13. B. The `wget` command retrieves a file from a Web (HTTP) server. Used with the `-r` option, `wget` retrieves an entire Web site, so option B is correct. Although `curl` can also be used to retrieve documents from Web sites, option A's syntax is incorrect, and `curl` isn't a good tool for retrieving an entire site recursively. Apache is a Web server, not a client, so option C is incorrect. Option D will launch the text-based `lynx` Web browser, enabling you to view the current site; but this action won't save the site for subsequent editing.
14. C. The `ftp` command to retrieve a file is `get`. (`receive` will also work, and multifile variants, such as `mget`, are also available.) The `cp` command is a Linux Bash command to copy a file, but it doesn't work in `ftp`. The `download` command is fictitious in this context. The `scp` command is a tool that's part of the Secure Shell (SSH) package.
15. D. The `lcd` command in `ftp` changes the local current directory; it does as the question specifies, so option D is correct. Option A sounds promising if you're familiar with Bash options; however, the change implemented in this way does not affect the already launched `ftp` instance, so option A is incorrect. The `cd` command in option B changes the current directory on the *server*, not on the *client*, as the question specifies. The `change-to` command in option C is fictitious.

16. D. The `curl` command retrieves a file using HTTP, FTP, or various other protocols and sends the file to standard output. Thus, option D will do as requested. (If the file is long, you might want to pipe it through `less`.) The `ftp` program can't connect to an HTTP (Web) server, and even if the computer were running an FTP server, option A wouldn't accomplish the specified goal. You could type **`telnet www.example.org 80`** to connect to the Web server and then type additional HTTP commands to retrieve the file; however, as stated, option B won't work. Although you can use `wget` to retrieve the file, `wget` won't display it directly on your screen, so option C is incorrect.
17. A. FTP supports both ASCII and binary modes. Transferring binary data (such as a JPEG graphics file) in ASCII mode will almost certainly corrupt it, as option A suggests. The difference between active and passive modes, referred to in option B, won't corrupt data. FTP is perfectly suitable as a tool for transferring graphics data, contrary to option C. Although encryption could result in the specified problem, as option D suggests, this seems implausible for files on a public FTP site. Furthermore, to decrypt files, you'd probably need the sender's *public* key, not a *private* key, as option D suggests.
18. B. The symptoms described can be a result of firewall settings blocking the current FTP transfer mode (active vs. passive). The `passive` command in the `ftp` program toggles between these two modes, so it may correct the problem. This isn't guaranteed, though. The `ascii` command of option A switches to ASCII mode, which alters text files for the client or server default; it's unlikely to help. The `mput` and `mget` commands of options C and D attempt to upload and download, respectively, files matching the `fix*` wildcard. This action is unlikely to be helpful.
19. A. The `mysqladmin` utility enables you to administer a MySQL server, including setting its administrative password. Option A presents the correct syntax to do this. Option B's `mysqlpasswd` utility is fictitious. Option C's `passwd` command will change the Linux login password for the `mysql` user. This action will have no effect on the MySQL server's administrative password, which is independent of the Linux account database. Likewise, option D will change the Samba password for the `mysql` user, which will have no effect on MySQL.
20. B. The `--host` (or `-h`) option to `mysql` tells it to connect to a remote server rather than the local one. The remaining options are all fictitious.



# Chapter 12

## Securing Linux

---

### THE FOLLOWING COMPTIA OBJECTIVES ARE COVERED IN THIS CHAPTER:

- ✓ 5.3 Explain the basics of SELinux (Running modes [Enabled, Disabled, Permissive]).
- ✓ 5.5 Explain the appropriate use of the following security related utilities (nmap, Wireshark, NESSUS, Snort, Tripwire).
- ✓ 5.6 Use checksum and file verification utilities (md5sum, sha1sum, gpg).
- ✓ 5.8 Explain the methods of authentication (PAM, LDAP, NIS, RADIUS, Two-factor authentication).



Sadly, security is a very important topic. Sloppy configuration, program bugs, user error, and other problems can result in a system compromise. Such a compromise can result in confidential data falling into the wrong hands, data loss, abuse of your resources (such as network connectivity) to malicious or even criminal ends, and so on. For these reasons, you should pay careful attention to security, configuring your system in as secure a way as possible.



Security isn't an all-or-nothing matter. No computer can be absolutely 100 percent secure—if nothing else, somebody might physically break in and steal the system. Rather, security comes in degrees, from very poor security up to very good security. You must decide where you want your system to fall on this continuum, trading off the benefits of improved security against the effort it takes to maintain that security, for both you and the system's users.

Many security topics are described in other chapters of this book. This chapter begins with a broad overview of security issues—where problems can exist. It then gets into the nitty-gritty details of how Linux authenticates users, since these details enable making changes with security implications. Using file encryption for transfers over the Internet is up next so that you can verify your correspondents' identities and encrypt private files and messages transferred via e-mail or other insecure media. This chapter then covers a set of high-security Linux tools that can be deployed in high-security environments. Next up is a description of tools and techniques you can use to review your system's overall level of security. Finally, this chapter looks at ways you can detect whether your system has been compromised.

## Sources of Security Vulnerability

Threats to system security are many and varied, and they're changing all the time. That is, bugs or other problems in specific programs are likely to be fixed soon after they're found, but new bugs or problems may be discovered the next day. Thus, a program-by-program listing of security problems is impractical in a book, although Web sites such as the Computer Emergency Response Team (CERT; <http://www.cert.org>) site do track known vulnerabilities. Instead of attempting to list all known problems, I describe security vulnerabilities in broad categories. These include physical access, stolen passwords, bugs in local (nonserver) programs, bugs in server programs, denial-of-service attacks, encryption issues, and humans.

## Physical Access Problems

The first broad category of security problem relates to physical access to the computer. In brief, if a miscreant has physical access to your computer, that person can do almost anything to it. Thus, controlling physical access to the computer is extremely important.

You may think of a system as secure because it has solid passwords and other protections. With a Linux emergency disk, though, somebody who gains physical access to your computer can boot into an environment that you don't control but that can still read and write all your important files, including sensitive files such as `/etc/shadow`. You can put up obstacles to easy access, such as using BIOS passwords. These require you to type a password before the computer boots at all. This protection can be overcome by resetting a jumper on the computer's motherboard, though, so a determined intruder can get around it. What's more, if data theft is the goal, the intruder can simply steal your hard disk, thus bypassing your BIOS password.

Very high-security installations may encrypt all the data on the hard disk to minimize the risk of data theft. This approach requires installing special software, such as loop-AES (<http://sourceforge.net/projects/loop-aes/>), dm-crypt (<http://www.saout.de/misc/dm-crypt/>), or True-Crypt (<http://www.truecrypt.org>).

Individual computers aren't the only elements for which physical access is a threat. Your local network infrastructure, if breached, can give intruders a way to access your local computers, bypassing the protections afforded by your router and firewall software. If you use a wired network exclusively, this risk is limited, but not nonexistent—somebody could break in or abuse access provided to visitors to plant a small computer on your network.



### Real World Scenario

#### Laptop Computer Security

Laptop computers present unusual security concerns. Such computers can be easily stolen, particularly if they're used in a portable manner rather than residing permanently in one secure room or building. If you use a laptop computer in a public space, it's also easy for other people to see what you're doing with it. Somebody who's up to no good might even be able to discern a password you type on a laptop in public, particularly if you type your password slowly.

You may want to consider using extreme measures, such as full-disk encryption, on laptop computers even when you wouldn't employ such tools on workstations or servers. At the very least, you should probably use different passwords on your laptops than on your workstations and servers.

The security picture for laptops isn't entirely bleak, though. Cases are occasionally reported of thieves being caught because of laptop features. For instance, one burglar (reported in <http://perens.com/works/articles/Burglar/>) was caught when he used a stolen laptop's camera to take photos of himself that were subsequently uploaded by an automatic backup utility to off-site storage maintained by the laptop's true owner!



With the explosion of wireless networking, the risk of internal network exposure is increased. Somebody can sit outside your building in a car and break into your local network. You can minimize this risk by employing Wi-Fi Protected Access 2 (WPA2) encryption, or at least the earlier WPA encryption. Avoid the still earlier Wired Equivalent Privacy (WEP) encryption, which is easily broken. For still greater protection, limit the things that can be done via a wireless link. For instance, you might isolate your wireless systems to their own subnet and use firewall rules to ensure they can do only those things they absolutely must be able to do. Another option is to employ additional encryption on top of the WPA2 encryption. You can use Secure Shell (SSH) to encrypt data transfers, for instance. You can even set up a virtual private network (VPN) for your Wi-Fi connections, which provides encryption and extra security over untrusted network links. Such configurations are very advanced, though, and I don't cover them in this book.

## Stolen Passwords

Passwords are an integral part of Linux user accounts, so if a password (with username) falls into the wrong hands, the password can give the intruder access to the computer. At first glance, this might not seem to be a huge problem—after all, ordinary user accounts have limited access. Unfortunately, if this access is combined with other problems (such as those described in “Local Program Bugs”), it can translate into a more severe root compromise. Even without this access, unauthorized use of local user accounts can be abused to send spam, to access other systems, to steal CPU time or other local resources, and to access whatever sensitive documents the user might be able to read.

Chapter 5, “Managing Users,” describes password security in greater detail. Key points include selecting good user passwords, changing passwords frequently, disabling unused accounts, and educating users about the risks of divulging their passwords to others (both directly and indirectly). Be sure that your users know to *never* give their passwords to others or to write them down. Attackers sometimes masquerade as system administrators or others in authority in an attempt to collect passwords, and they've even been known to go rummaging through trash to locate discarded passwords or other sensitive data.



The root password is particularly sensitive. Thus, you should be extremely diligent in selecting a good root password and in protecting it from compromise.

## Local Program Bugs

For the most part, bugs in computer programs are considered annoyances. When a spreadsheet program crashes, you may sigh in frustration or scream in rage, but chances are the computer on which the program is running won't be damaged or compromised by this crash. Some program bugs, though, are more serious, because they can be abused to give an attacker increased access to the computer.



Most programs have limited access to truly sensitive files, data, and hardware. Thus, most local program bugs are unlikely to be useful to attackers. The main risk comes from local programs that run with enhanced privileges—that is, those that enable a set user ID (SUID) or set group ID (SGID) bit. These features enable a program to run with the privileges associated with the program's owner or group, respectively. As described in Chapter 3, "Managing Processes and Editing Files," SUID and SGID bits are risky, in part because bugs in these programs can turn into accesses by another user. As the other user is often *root*, bugs in SUID programs in particular might be abused to alter other files, including configuration files such as */etc/passwd*. Thus, a clever attacker (or a not-so-clever attacker who uses attack scripts created by others) can, at least in principle, abuse local program bugs to acquire *root* privileges, effectively taking over the computer.



When an attacker gains *root* privileges on a computer, that system is sometimes said to have been *rooted*.

Because of the security implications of SUID and SGID programs, you may want to check your system to learn what programs set these bits. You can do so with the *find* command, which is described in more detail in Chapter 2, "Using Text-Mode Commands."

```
find / -perm +6000 -type f
```

The preceding command finds all the files on the computer (including any mounted removable media or network filesystems) that have their SUID or SGID bits set. To search for SUID files alone, change *+6000* to *+4000*; to search for SGID bits alone, change *+6000* to *+2000*. The *-type f* parameter is important to keep directories from showing up in the output; this parameter restricts the search to normal files. I've shown this example using the *root* prompt (*#*) because only *root* is likely to be able to read all the files and directories on the computer. Although this command can be run as an ordinary user, it will return several *permission denied* errors and might miss some files as a result.

Local program bugs can be exploited only by people who have access to local programs. Thus, they might at first seem to be of little interest if the computer has no local users aside from administrative staff or if you're certain local users can be trusted. Unfortunately, these bugs can sometimes be exploited should an ordinary account be compromised (say, through a stolen password). Thus, you should be concerned with such bugs even on servers with no local users.

The main defense against local program bugs is keeping your system up-to-date. You should use the Advanced Package Tools (APT), Yum, YaST2, or any other tools available to you to keep your packages up-to-date—or at least, those that have been updated to fix security bugs.

## Server Bugs

Server programs, like local user programs, can contain bugs. Like local user programs, these bugs are most serious when the program is run as *root*. Most servers don't use SUID

bits to run as `root`, though; they're launched via SysV startup scripts or a super server. Thus, there's no `find` command to locate server programs that will run as `root`. Instead, you must review your SysV and super server configurations (as described in Chapter 4, "Managing System Services"). Auditing your system to locate running servers can also be helpful, as described later in "Checking for Open Ports." As with local programs, you should also be sure to keep all your server programs up-to-date or at least update them when you hear of security issues that have been fixed.

Unlike local user program bugs, bugs in servers can cause security breaches even when the system has no local user accounts. For instance, a bug in a Web server could, at least theoretically, enable a cracker to run arbitrary code as `root`. That code could create a `root`-equivalent account and launch a Telnet server, enabling the cracker to gain full `root`-level shell access to the computer.

## Denial-of-Service Attacks

A *denial-of-service* (DoS) attack is unusual because it needn't involve an actual security breach on your system (although it might). The term applies to any type of attack that denies you the use of your equipment. One common type of DoS attack is a *distributed denial-of-service* (DDoS) attack, in which the attacker uses many computers (typically hijacked in one way or another long before) to flood the victim's computer with useless data packets. The result is that the victim's computer cannot send or receive real data over the network. For a Web server, mail server, or other computer that's used mainly as a network server, the effect is as devastating as if the attacker had broken into the computer and shut it down.

Other types of DoS attack do exist. For instance, if a server program crashes upon receiving certain input, an attacker could simply send that input to the server, thus causing it to crash. The attacker hasn't broken into the computer, much less rooted it, but the disruption can be quite severe. Particularly to large Internet service providers (ISPs), spam can look a lot like a DoS attack—by consuming network resources, a spike in spam can cause disruption of the ISP's normal operation.

Some DoS attacks can be guarded against by keeping your system up-to-date. In particular, DoS attacks that target bugs in software can be thwarted by fixing those bugs. Other DoS attacks, though, require coordination between you and your ISP. If you find that a server is under a DDoS attack, for instance, you might not be able to do much about it on your server; you must work with your ISP to identify the sources of the attack or some other way to "fingerprint" the relevant packets and drop them before they're sent to your system. That said, some types of firewall configuration can mitigate the effects of a DDoS attack, either by keeping the traffic off of your local network or by causing the server computer to ignore the packets rather than reply to them. (Chapter 9, "Configuring Advanced Networking," describes `iptables`, the Linux firewall tool.)

## Encryption Issues

Another type of vulnerability relates to encryption—or more precisely, the lack thereof. Many network protocols send data in unencrypted form. The Simple Mail Transfer Protocol

(SMTP), the Hypertext Transfer Protocol (HTTP), Telnet, the File Transfer Protocol (FTP), and many others do not encrypt data. In some cases, users can encrypt data to be sent via these protocols. For instance, e-mail users can employ the GNU Privacy Guard (GPG; <http://www.gnupg.org>) to encrypt their e-mail messages, but the protocols themselves are unencrypted and often carry unencrypted data. This fact can become a threat because unencrypted data can be intercepted and read on any intervening system, and sometimes on computers on the same network as the source or destination. If sensitive data, such as passwords or credit card numbers, are passed over these unencrypted protocols, the result is a risk that the sensitive information will fall into the wrong hands.

The solution to this problem is to use encrypted protocols or to add encryption whenever possible. For instance, you can retire a Telnet or FTP server program in favor of SSH, which can do the same job as both Telnet and FTP but using encryption. Some protocols have encrypted variants, such as the secure HTTP variant, HTTPS, which uses Secure Sockets Layer (SSL) encryption. (Most Web pages that ask for credit card numbers or other sensitive data use HTTPS, as indicated by the `https://` in the URL. Most browsers also display a closed padlock, another security icon, or a URL in a distinctive color when accessing a site that employs encryption.)

Not all encryption is equal, though. Encryption methods vary in many ways, one of the most important being the length of encryption keys. These are numbers that are used to mathematically scramble the data being sent. Without the original key or a key that's matched to it, the data can't be unscrambled. All other things being equal, longer keys are superior to shorter ones. Precisely how long your key should be depends on the protocol, the type of data you're transmitting, and how time sensitive the information is. Breaking a key might take a few minutes, a few months, or decades. As computers speed up, the time to break encryption goes down. Currently, most encryption tools support 128-bit (16-byte) or larger keys.

## The Human Element

People can render even the best security plans useless, either through malice or through ignorance. One scenario involves *social engineering*—an attacker simply asks a legitimate user for a password or to otherwise bypass a security measure. Of course, the attacker refrains from twirling a long mustache while making the request. Typically, the social engineer poses as a system administrator or some other authority figure and asks for the information in a way that seems plausible. The attacker may claim that a password database must be reinitialized, for instance.



A particular type of social engineering known as *phishing* has become quite common on the Internet at large. Phishing involves sending bogus e-mail or setting up fake Web sites that lure unsuspecting individuals into divulging sensitive financial or other information.

Users can also create security problems by leaving sensitive doors unlocked, by running poorly designed scripts on servers, by installing unnecessary server programs, and so on.

Note that this list of activities includes some that are likely to be done by system administrators, as well as by ordinary users. Indeed, configuration errors are a major source of security breaches—for whatever reason, too many administrators don’t take the necessary steps to secure their computers.

The main way to guard against human errors that lead to compromise is education. Reading this chapter, as well as the other security advice in this book, is a good start for a Linux system administrator. Keeping up-to-date by reading security newsgroups and Web sites (such as the CERT site, <http://www.cert.org>) is another big way to help. Educate your users about the presence of social engineers and phishing.

## Authenticating Users

Because user accounts, passwords, and related issues are so important in Linux security, I describe these issues in some detail. Linux uses two subsystems, the Name Service Switch (NSS) and Pluggable Authentication Modules (PAM) to manage accounts and authenticate users, respectively. Understanding how these systems interact will enable you to modify how they function to improve security or to enable more flexible network authentication systems.

### Understanding How Linux Authenticates Users

As described in Chapter 5, Linux stores its account information in two files: basic account information is held in `/etc/passwd`, while passwords and ancillary account information reside in `/etc/shadow`. This is true for most stock Linux installations, but it’s also a simplification of a larger truth. In fact, Linux employs two subsystems to manage accounts and authenticate users: NSS and PAM.

NSS is a name-management tool; it manages various names in Linux, such as computer hostnames, network protocol names, and usernames. When you use a command that references a username, such as `usermod` or `passwd`, NSS swings into action; it verifies that the username you reference exists (or doesn’t exist) and translates the username into a user ID (UID) number. NSS does the same thing for group names.

NSS is configurable. It’s controlled via the `/etc/nsswitch.conf` file. Of particular interest to Linux authentication are the following lines from this file:

```
passwd: compat
group: compat
shadow: compat
```

These lines tell Linux what it should use for information that’s normally associated with `/etc/passwd`, `/etc/group`, and `/etc/shadow`, respectively. The `compat` keyword tells NSS to use the ordinary files, as described in Chapter 5. (Some distributions use the keyword `files` instead of `compat` in this place.) You can change this keyword, or add others, to have NSS use other tools instead of or in addition to the standard disk files.

Verifying that an account exists is the first step in authentication. The second step is to check the password that the user types using an authentication tool. In Linux, this task is handled via PAM. The idea behind PAM is that every tool that requires authentication has its own needs and so can be configured individually. This is done via files in `/etc/pam.d`, as described in “Configuring PAM.” PAM enables you to set up different rules for authenticating users for each login method—local text-mode console logins, local GUI console logins, remote logins via SSH, e-mail retrieval via POP or IMAP, and so on. This feature can be useful because you can set different conditions and actions. For instance, you might want to ensure that users have valid home directories for console or remote text-mode or GUI logins, but this requirement may not be important for users who only retrieve e-mail via a POP mail server.

## Configuring PAM

Unfortunately, PAM configurations vary substantially from one distribution to another. Typically, `/etc/pam.d` contains a large number of files, most of which are named after particular servers or other login tools, such as `gdm`, `login`, `passwd`, `sshd`, and `su`. (Very old distributions used the `/etc/pam.conf` file to configure PAM, but this style of PAM configuration has long since gone the way of the dodo.) To better understand PAM, consider a typical PAM configuration file, from `/etc/pam.d/login` on a Fedora system, shown in Listing 12.1.

### Listing 12.1: A Typical PAM Configuration File

```

#%PAM-1.0
auth [user_unknown=ignore success=ok ignore=ignore default=bad]
pam_securetty.so
auth include system-auth
account required pam_nologin.so
account include system-auth
password include system-auth
pam_selinux.so close should be the first session rule
session required pam_selinux.so close
session required pam_loginuid.so
session optional pam_console.so
pam_selinux.so open should only be followed by sessions to be executed
in the user context
session required pam_selinux.so open
session required pam_namespace.so
session optional pam_keyinit.so force revoke
session include system-auth
session optional pam_ck_connector.so

```

A PAM configuration for a specific service includes four stacks, which are sets of PAM modules that are called in a specific order to perform particular subtasks associated with authentication. Listing 12.1 shows four stacks, as identified by the management group keywords in the first column of the noncomment lines: `auth`, `account`, `password`, and `session`. Each management group defines how PAM controls particular authentication tasks: `auth` verifies passwords, `account` manages nonpassword access rules, `password` handles password maintenance, and `session` handles bookkeeping when users log in or out.

The second column in a PAM configuration file specifies how success or failure of each module is to affect the authentication task. For instance, `required` indicates that the module must succeed in its task, whereas `optional` means that the task can succeed or fail with no particular repercussions. The `include` keyword means that another file in `/etc/pam.d` is loaded at this point in the stack. (Some distributions use a directive called `@include` in the first column rather than an `include` directive in the second column for this purpose.) Including other files provides a way to easily modify the configuration of several authentication services: you can modify only the included file rather than the files for every tool on the system.

The final column in a PAM configuration file is the name of the module, sometimes with associated options. This is where configuring PAM gets tricky; lots of modules are available, and some add-on packages ship with their own tools. For instance, Samba ships with the Winbind tool for authenticating users against a Windows network account database, and Winbind consists, in part, of a PAM module that you add to an account stack.

Wholesale changes to PAM configuration frequently involve the general-purpose authentication stack—the one that’s referred to by any `include` directives in other files, such as `system-auth` in Listing 12.1. You can add or change a module to affect most of the relevant authentication tools.



Changing your PAM configuration is risky. If you get something wrong, you can end up unable to log into the computer. If this happens, you may need to use an emergency boot disk to restore your original files. *Always* make backups of any files you change in `/etc/pam.d` so you can do this. You should also leave a text-mode root console login running when you make your changes. That way, if you test your changes and find they block all access, you’ll be able to revert to your original configuration without booting your emergency disk.

Unfortunately, changing PAM stacks is potentially tricky. Every distribution does things slightly differently, so a change that works well on one distribution may not work at all on another. As a general rule, if you want to add a new authentication tool, you must add a line with the new module to the `auth` and `account` stacks. This line is typically typed `sufficient` and appears immediately before or after the `pam_unix.so` module. In some cases you may need to alter the `pam_unix.so` line or make other changes.

When using network or other unusual authentication methods, you’ll still need to create home directories for these users. If you want this to be done automatically, try adding the following line to the end of the `session` stack:

```
session required pam_mkhomedir.so skel=/etc/skel umask=0022
```

This PAM module creates a home directory for users who don't already have one. This option can be very handy when authenticating against Winbind, NIS, LDAP, or other network servers that maintain account databases but that don't create home directories on the Linux systems that use the authentication server.

## Using Network Authentication

One means of extending Linux authentication is to enable a network server to hold Linux account information. Chapter 10, “Configuring Network Servers I,” and the preceding section, “Configuring PAM,” mentioned Winbind, which enables you to take advantage of a Windows domain server to provide you with account information. A couple more approaches are the *Network Information Service (NIS)* and the *Remote Authentication Dial In User Service (RADIUS)* tools.

### Using NIS

NIS is a protocol that's designed to simplify user authentication and related services on a network of multiple Unix or Linux systems. There are several variants of NIS, such as NIS+ and NIS YP and Switch (NYS). The original NIS was once called Yellow Pages (YP), but that's a registered trademark in some areas, so the name was changed. Nonetheless, most NIS utilities and configuration files still include `yp` in their names.

Some distributions let you configure NIS during system installation. You may be required to enter the NIS domain name (which may be different from your DNS domain name) and the address of the NIS server. If you want to use NIS after installing the OS, your task is a bit trickier. Your distribution might provide GUI tools to help the process, or you might need to configure the system manually. You should begin by installing the NIS packages. For an NIS client, the `ypbind` package is the most important one, but on most distributions it depends on other packages, such as `yp-tools`.



NIS uses both clients and servers. The NIS server holds network account information, and the NIS clients use that information to authenticate users. This section describes the basics of NIS client configuration and the use of certain tools for NIS account maintenance. To learn about NIS server configuration, consult its documentation or a book on the subject, such as Hal Stern, Mike Eisler, and Ricardo Labiaga's *Managing NFS and NIS, 2nd Edition* (O'Reilly, 2001).

The `ypbind` NIS package's main configuration file is `/etc/yp.conf`. This file's primary purpose is to point the NIS tools at an NIS server. The default file normally presents several possible ways to do this in comments. These methods differ in the amount of information you as an administrator have. For instance, if you know the name of your NIS domain and the name of the NIS server, you might use this format:

```
domain NISDOMAIN server NISSERVER
```

At the opposite extreme is a line that contains a single word: **broadcast**. This tells the tools to send out a broadcast query for a suitable NIS server. You should consult your network administrator to learn what option is best for your network.

Once you've told the NIS tools about your server, you must also configure Linux to use NIS. This can be accomplished by editing the `/etc/nsswitch.conf` file, which tells Linux what tools to use for name resolution, account information, and so on. A configuration that relies heavily on NIS is shown in Listing 12.2. If you want to use NIS only for user authentication, you can simply change the `passwd`, `group`, and `shadow` lines to add `nis` to the existing options.

**Listing 12.2:** Sample `/etc/nsswitch.conf` File for NIS

```
passwd: compat
group: compat
For libc5, you must use shadow: files nis
shadow: compat

passwd_compat: nis
group_compat: nis
shadow_compat: nis

hosts: nis files dns

services: nis [NOTFOUND=return] files
networks: nis [NOTFOUND=return] files
protocols: nis [NOTFOUND=return] files
rpc: nis [NOTFOUND=return] files
ethers: nis [NOTFOUND=return] files
netmasks: nis [NOTFOUND=return] files
netgroup: nis
bootparams: nis [NOTFOUND=return] files
publickey: nis [NOTFOUND=return] files
automount: files
aliases: nis [NOTFOUND=return] files
```

Once this is set up, you should start or restart the `ypbind` daemon, which must be running at all times on the NIS client. Client-side tools contact this daemon for information that's normally stored in `/etc/passwd` and elsewhere, the `ypbind` daemon contacts the NIS server, and the NIS server delivers the requested information. Normally, you start `ypbind` via its SysV startup script.

Once NIS is up and running, you can manage the system with an assortment of commands whose names begin with `yp`. For instance, `yppasswd` changes a password much as `passwd` does, but `yppasswd` changes the password on the NIS server. On the server system, `ypinit` initializes the user account database.



## Using LDAP

The Lightweight Directory Access Protocol (LDAP) is similar to NIS in some ways, but it's more sophisticated and general in purpose. A *directory*, in the context of LDAP, is a type of database. An LDAP server can hold data in its directories on any number of things, including, of relevance to this chapter's topic, account information for Linux computers.

Although configuring a Linux system as an LDAP server is well beyond the scope of this chapter, configuring Linux as an LDAP client for authentication is much simpler. In brief, you must configure both NSS and PAM to use LDAP in addition to (or instead of) the regular local account files.

You must first install the LDAP authentication packages. Details differ from one distribution to another, but package names are likely to be `pam_ldap` and `nss_ldap`, or something similar. (Sometimes `lib` comes at the start of these package names.) Locate and install these packages using your distribution's package tools.

With the packages installed, you should edit the `/etc/ldap.conf` file. Look for the `host` and `base` options in this file, and set them appropriately for your network:

```
host 172.24.21.102
base dc=luna,dc=edu
```

The `host` line specifies the hostname or IP address of the LDAP server, and `base` specifies the base of the LDAP directory tree in which the account information resides. If you don't know these values, ask the person who maintains your LDAP server.

To configure NSS to use LDAP, you must edit `/etc/nsswitch.conf`, as described earlier in "Using NIS." Chances are adding `ldap` to the ends of the `passwd`, `shadow`, and `group` lines will do what you want:

```
passwd: compat ldap
shadow: compat ldap
group: compat ldap
```

PAM configuration is trickier, since the details vary from one distribution to another, as described earlier in "Configuring PAM." You should add a reference to `pam_ldap.so` to both the `auth` and `account` stacks for any service that should use LDAP:

```
auth sufficient pam_ldap.so try_first_pass
account sufficient pam_ldap.so
```

With these changes in place, users should be able to log in using accounts that are defined on the LDAP server.

## Understanding RADIUS

RADIUS is a network authentication protocol that's designed as a single authentication tool for several types of network access. A RADIUS server frequently has network portals as clients—dial-up network servers, wireless access points (WAPs), and so on. Each of these clients needs to authenticate users of the network as a whole, and to simplify overall network

configuration, a RADIUS server is configured to centralize the authentication tasks associated with the various access methods.

RADIUS may in turn use another network tool, such as an LDAP server, to hold its authentication data. The result can be a complex configuration. Despite this complexity, RADIUS can be worth using on large networks with multiple methods of access that require authentication.

You can run a RADIUS server, such as FreeRADIUS (<http://freeradius.org>), on Linux. Because of the complexity and sensitivity of RADIUS configuration, I don't describe it in any detail here. For the purpose of the Linux+ exam, you should understand what RADIUS is. If you need to actually install and use it, I recommend you consult its documentation.

## Using Two-Factor Authentication

Ordinary Linux authentication involves asking the user for a username and a password. This approach is fine for most installations; however, in some cases, something stronger is required. In these cases, you may resort to *two-factor authentication*. This term refers to any authentication system that uses two different types of authentication data. These types include the following:

- Information the user has, such as a username, a password, or the user's favorite color
- Possessions the user has, such as a key or ID badge
- Features that describe the user, such as fingerprints or retina prints

The first category of information is obviously the easiest to obtain with Linux. You can use a physical key on a computer to provide a second authentication factor, but of course such obstacles are easy to overcome if an intruder has stolen the computer. Biometrics, such as fingerprint scanners, are used in some high-security installations to provide additional security. Although such devices can be obtained from various sources, you must track down appropriate Linux software to integrate them into a PAM authentication stack.

## Using GNU Privacy Guard (GPG)

Sometimes you may want to encrypt e-mail messages or files sent to another person via some other means. E-mail was never designed as a secure data transfer tool, and most e-mail messages pass through several e-mail servers and network routers. A compromise at any one of these points enables a cracker to sniff e-mail traffic and extract sensitive data, such as credit card or Social Security numbers. Encrypting your e-mail keeps such details private.

The usual tool for encrypting e-mail is the GNU Privacy Guard (GPG or GnuPG; <http://www.gnupg.org>) package. This package is an open source reimplementation of the proprietary Pretty Good Privacy (PGP). In addition to encrypting entire messages, GPG enables you to digitally "sign" messages. Used in this way, messages can be read by recipients who lack the GPG software or appropriate keys, but those who have these tools can verify that the contents haven't been tampered with.

## Generating and Importing Keys

To begin using GPG, you should first install the software. Chances are your distribution includes it as a standard package, so you can install it that way. Once this is done, you must generate keys. GPG keys are conceptually similar to SSH keys: you need a private key (aka a secret key) and a public key. As the names imply, the private key is kept private, but the public key is publicly available. You can sign a message with your private key, and readers can verify it with your public key; or you can encrypt a message with another user's public key, and it can be decrypted only with that user's private key.

To generate keys, you use the `gpg` program with its `--gen-key` option:

```
$ gpg --gen-key
```

The program will ask you a series of questions. In most cases, answering with the defaults should work well, although you may have to type in your full name and e-mail address. The keys are stored in the `~/.gnupg` directory.

Once you've generated your keys, you can *export* your public key:

```
$ gpg --export name > gpg.pub
```

This command saves the public key associated with *name* in the file `gpg.pub`. You can use your e-mail address as *name*. (If you create additional public keys or add others' public keys, you can specify their names to export those keys.) You can then make your key available to others so that they may encrypt e-mail messages sent to you or verify your signed messages. Adding the `--armor` option produces ASCII output, which may be preferable if you intend to e-mail the public key. You can make the file accessible on your Web site, transfer it as an e-mail attachment, or distribute it in various other ways.

To encrypt e-mail you send to others, you must obtain their public keys. Ask your correspondents how to obtain them. Once you've done so, you can add their keys to your *keyring* (that is, the set of keys GPG maintains):

```
$ gpg --import filename
```

This command adds *filename* to your set of public keys belonging to other people.



Although public keys are, by definition, public, there are security concerns relating to them. Specifically, you should be sure you use a *legitimate* public key. Hypothetically, a miscreant could publish a fake public key in order to obtain sensitive communications or fake a signed e-mail. For instance, George might distribute a fake GPG public key, claiming it to be from Harold. George could then either sign messages that appear to be from Harold or intercept e-mail sent to Harold that was encrypted using the fake key. Thus, you should use as secure a communication method as possible to distribute your public key and to receive public keys from others.

Once you've created your own key and, perhaps, imported keys from others, you can see what keys are available by using the `--list-keys` option to `gpg`:

```
$ gpg --list-keys
/home/gjones/.gnupg/pubring.gpg

pub 1024D/190EDB2E 2008-09-05
uid George A. Jones <gjones@example.com>
sub 2048g/0D657AC8 2008-09-05

pub 1024D/A8B2061A 2008-09-05
uid Jennie Martin <jennie@luna.edu>
sub 2048g/4F33EF6B 2008-09-05
```

The `uid` lines contain identifiers you'll use when encrypting or decrypting data, so you should pay particular attention to that information.

## Encrypting and Decrypting Data

To encrypt data, you use `gpg` with its `--out` and `--encrypt` options and, optionally, `--recipient` and `--armor`:

```
$ gpg --out encrypted-file --recipient uid --armor --encrypt original-file
```

You can use the UID from a `gpg --list-keys` output, or just the e-mail address portion, as the `uid` in this command. If you haven't signed the recipient's key, you'll have to verify that you want to use that key. The result is a new file, *encrypted-file*, which holds an encrypted version of *original-file*. If you omit the `--armor` option, the resulting file is a binary file; if you send it as e-mail, you'll need to send it as an attachment or otherwise encode it for transmission over the text-based e-mail system. If you include the `--armor` option, the output is ASCII, so you can cut and paste the encrypted message into an e-mail or send it as an attachment.

If you receive a message or file that was encrypted with your public key, you can reverse the encryption by using the `--decrypt` option:

```
$ gpg --out decrypted-file --decrypt encrypted-file
```

You'll be asked to enter your passphrase. The result should be a decrypted version of the original file.

In practice, GPG can be even easier to use than this description may make you think. GPG is primarily used to secure and verify e-mail, so most Linux e-mail clients provide GPG interfaces. These programs call `gpg` with appropriate options to encrypt, sign, or decrypt messages. Details vary from one e-mail client to another, so you should consult your program's documentation for details.

## Signing Messages and Verifying Signatures

As noted earlier, GPG can be used to sign messages so that recipients know they come from you. To do so, use the `--sign` or `--clearsign` option to `gpg`:

```
$ gpg --clearsign original-file
```

The `--sign` option creates a new file with the same name as the original, but with `.gpg` appended to the filename. This file is encrypted using your private key so that it may be decrypted only with your public key. This means that anybody with your public key may read the message, but anybody who can read it knows it's from you. The `--clearsign` option works similarly, but it leaves the message text unencrypted and merely adds an encrypted signature that can be verified only by using your public key. The `--clearsign` option creates a file with a name that ends in `.asc`.

If you receive a signed message, you can verify the signature using the `--verify` option to `gpg`:

```
$ gpg --verify received-file
```

If any of the keys in your keyring can decode the message or verify the signature, `gpg` displays a `Good signature` message. To read a message that was encrypted via the `--sign` option, you must decrypt the message via the `--decrypt` option, as described earlier.

## SELinux

Security Enhanced Linux (SELinux) is a set of extensions to Linux's usual security model designed to improve the overall security of the system. Using SELinux, you can restrict the actions that individual users can perform in a very fine-tuned manner. This can be helpful in enabling some users to perform just a subset of normal administrative tasks.

### Principles of SELinux

SELinux extends the Linux security system by introducing the concept of a *mandatory access control (MAC)*, which is a set of rules that restricts what users, including `root`, may do with files. Under SELinux, every user is part of a *domain*, which is a set of users who may be granted access to each others' files. SELinux also enables multiple administrative users, who may be in different domains. Thus, you could have one administrator who's responsible for maintaining Apache and another who's responsible for maintaining X. If these administrators are in different domains, the Apache administrator can't damage the X configuration, even accidentally. This configuration has obvious security benefits on a system with multiple administrators.

In addition to these features, processes are assigned to *contexts*, which restrict what the process may do. In practice, this means that the Apache process (for example) can access

Apache-related files only. Thus, if a bug exists in Apache, the server's context limits the damage that Apache can do, either accidentally or if the bug is abused by a cracker. The Apache context is unlikely to grant it access to `/etc/passwd`, for instance.

SELinux requires support in the Linux kernel and in many programs and support tools outside of the kernel. If you want to use it, you should first ensure that your kernel includes SELinux support. You can then install relevant SELinux packages, whose names are likely to start with the string `selinux`. You may need to install several packages, including one or more *policy* packages, which include the rules that define MACs, domains, contexts, and so on, to make a working SELinux system.



SELinux can be confusing, in part because it reuses so many terms. *MAC* can also refer to media access control, a type of low-level network address; and the word *domain* is used in many other network contexts, such as the Domain Name System (DNS) or a Microsoft Windows domain.

In practice, SELinux can be very complex and confusing. It also degrades the system's performance slightly because the kernel must perform additional security checks. If the default policies provided with your distribution aren't suitable for your installation, you'll have a hard time modifying them—and this task is complex enough that I can't cover it in this book. (The Linux+ exam also doesn't cover SELinux policy configuration.)

## Configuring SELinux Running Modes

Once an SELinux kernel is running and the support tools are installed, SELinux will run in one of three modes:

**Disabled** The SELinux extensions are turned off completely.

**Permissive** The SELinux extensions are turned on enough to log results, but the system doesn't actually restrict users' actions. This can be a good mode for testing your SELinux configuration prior to bringing a system completely online.

**Enabled** The SELinux extensions are fully operational, restricting accesses based on your SELinux policies.

You can switch your SELinux modes in several ways. One is to use the `/selinux/enforce` file, which is part of an SELinux-specific virtual filesystem (`/selinux`). You can store a value of 0 or 1 in this file to switch between permissive and enabled modes, respectively:

```
echo 1 > /selinux/enforce
```



If SELinux is enabled, you'll need to be in the `sysadm_r` role to modify files in `/selinux`. You can use the `newrole` command to switch roles, as in `newrole -r sysadm_r`.

Some distributions provide a `setenforce` command that accomplishes the same task:

```
setenforce 1
```

You can permanently set the running mode on some distributions, such as Fedora and Red Hat, by editing `/etc/selinux/config`. Look for the `SELINUX` option, and set it to `enforcing`, `permissive`, or `disabled`, as in the following example:

```
SELINUX=enforcing
```

Some distributions don't provide the `/etc/selinux/config` file. On these distributions, editing your boot loader's boot options will usually enable or disable SELinux. In particular, add `selinux=0` to disable SELinux or `selinux=1` to enable it. In GRUB (`/boot/grub/menu.lst` or `/boot/grub/grub.conf`), this might look like the following:

```
kernel /boot/bzImage-2.6.19 ro root=/dev/hda1 nousb selinux=0
```

## Security Auditing

From time to time, you should check your system for suspicious configurations. Such security auditing can detect intrusions that might have slipped past other detection tools and procedures. It can also catch sloppy configurations that might lead to trouble in the future. Examples of things you should check are scanning for open ports, reviewing your local accounts, and reviewing the installed files and packages.

### Checking for Open Ports

*Open ports* are those that respond to connection attempts—that is, servers are running on the ports. Ideally, the only open ports on a system will be those associated with servers you intend to run. Sometimes, though, a port will be open because of an accidental misconfiguration or because a cracker has broken into your system. Thus, scanning for open ports is an important security precaution. Two methods of spotting unnecessary servers are to use local network activity tools and to use a network scanner.

### Using Local Network Activity Tools

One tool that can be helpful in spotting stray servers is `netstat`. This program is the Swiss Army knife of network status tools; it provides many different options and output formats to deliver information on routing tables, interface statistics, and so on. For purposes of spotting unnecessary servers, you can use `netstat` with its `-a` and `-p` options, as shown here:

```
netstat -ap
```

```
Active Internet connections (servers and established)
```

| Proto            | Recv-Q | Send-Q | Local Address          | Foreign Address        | State       |
|------------------|--------|--------|------------------------|------------------------|-------------|
| PID/Program name |        |        |                        |                        |             |
| tcp              | 0      | 0      | *:ftp                  | *:*                    | LISTEN      |
| 690/inetd        |        |        |                        |                        |             |
| tcp              | 0      | 0      | tee1a.rodbooks.com:ssh | nessus.rodbooks.:39361 | ESTABLISHED |
| 787/sshd         |        |        |                        |                        |             |



I’ve trimmed most of the entries from the preceding netstat output to make it manageable as an example.

The Local Address and Foreign Address columns specify the local and remote addresses, including both the hostname or IP address and the port number or associated name from /etc/services. The first of the two entries shown here isn’t actively connected, so the local address, the foreign address, and the port number are all listed as asterisks (\*). This entry does specify the local port, though—ftp. This line indicates that a server is running on the ftp port (TCP port 21). The State column specifies that the server is listening for a connection. The final column in this output, under the PID/Program name heading, indicates that the process with a process ID (PID) of 690 is using this port. In this case, it’s inetd.

The second output line indicates that a connection has been established between tee1a.rodbooks.com and nessus.rodbooks.com (the second hostname is truncated). The local system (tee1a) is using the ssh port (TCP port 22), and port 39361 is used on the client (nessus). The process that’s handling this connection on the local system is sshd, running as PID 787.

It may take some time to peruse the output of netstat, but doing so will leave you with a much-improved understanding of your system’s network connections. If you spot servers listening for connections that you didn’t realize were active, you should investigate the matter further. Some servers may be innocent or even necessary. Others may be pointless security risks.



When you use the -p option to obtain the name and PID of the process using a port, the netstat output is wider than 80 columns. You may want to open an extra-wide xterm window to handle this output, or you may want to redirect it to a file that you can study in a text editor capable of displaying more than 80 columns. To quickly spot servers listening for connections, pipe the output through a grep LISTEN command to filter on the listening state. The result will show all servers that are listening for connections, omitting client connections and specific server instances that are already connected to clients.

## Using Remote Network Scanners

Network scanners, such as Nmap (<http://www.insecure.org/nmap/>) or Nessus (<http://www.nessus.org>), can scan for open ports on the local computer or on other computers. The



more sophisticated scanners, including Nessus, will check for known vulnerabilities, so they can tell you whether a server might be compromised should you decide to leave it running.



Network scanners are used by crackers for locating likely target systems, as well as by network administrators for legitimate purposes. Many organizations have policies forbidding the use of network scanners except under specific conditions. Therefore, you should check these policies and obtain explicit permission, signed and in writing, to perform a network scan. Failure to do so could cost you your job or even result in criminal charges, even if your intentions are honorable.

Nmap can perform a basic check for open ports. Pass the `-sT` parameter and the name of the target system to it, as shown here:

```
$ nmap -sT teela.rodsbooks.com
```

```
Starting nmap V. 3.55 (www.insecure.org/nmap/) at 2004-12-21 12:11 EDT
```

```
Interesting ports on teela.rodsbooks.com (192.168.1.2):
```

```
(The 1581 ports scanned but not shown below are in state: closed)
```

| Port   | State | Service |
|--------|-------|---------|
| 21/tcp | open  | ftp     |
| 22/tcp | open  | ssh     |



As with the output of `netstat` shown in “Using Local Network Activity Tools,” the preceding output for Nmap has been trimmed for brevity’s sake.

This output shows two open ports—21 and 22, used by `ftp` and `ssh`, respectively. If you weren’t aware that these ports were active, you should log into the scanned system and investigate further, using `netstat` or `ps` to locate the programs using these ports, and if desired, shut them down. The `-sT` option specifies a scan of TCP ports. A few servers, though, run on UDP ports, so you need to scan them by typing `nmap -sU hostname`. (This usage requires root privileges, unlike scanning TCP ports.)

Nmap is capable of more sophisticated scans, including “stealth” scans that aren’t likely to be noticed by most types of firewalls, ping scans to detect which hosts are active, and more. The Nmap man page provides details.

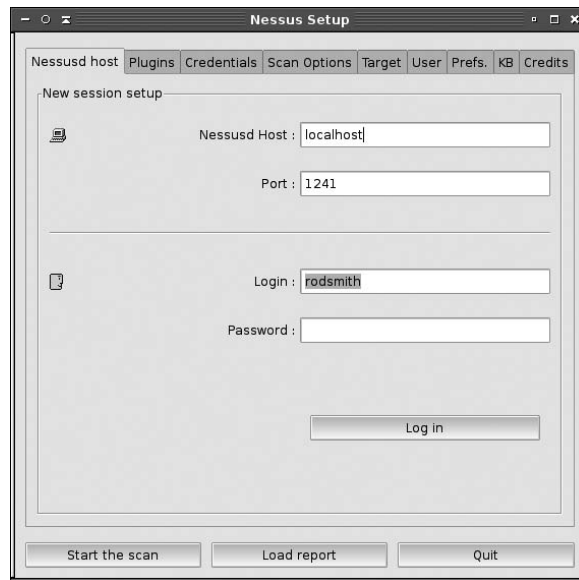
Nessus is a more sophisticated network scanner than Nmap. It comes as separate client and server components; the client enables you to control the server, which does the actual work. You’ll typically need to install the two parts separately, so check your distribution’s package manager to find and install them both. You can install the server on one system and the client on another, if you like.

Once Nessus is installed, you must create a Nessus account. This is an account in Nessus itself, not a regular Linux account. Type `nessus-adduser` on the Nessus server system. The

program will ask a series of questions. Answer them, and when the program finishes, you'll be able to run the Nessus client to perform network scans.

To run the client, type **nessus** on the client system. (You can run the Nessus client as an ordinary user). The result resembles Figure 12.1. You should begin with the Nessusd Host tab; enter the hostname of the Nessus server in the Nessusd Host field. You can then enter the username and password for the account you created with `nessusd-adduser`. When you click Log In, the program presents a few dialog boxes with security information. That done, the program switches to the Plugins tab, which you can use to enable or disable various tests. The remaining tabs provide a wide array of advanced options for controlling the port scan. Peruse these options and change them as you see fit, but if you're not sure what something does, the default is a good starting point.

**FIGURE 12.1** Nessus provides a GUI to facilitate the performance of complex network scans.

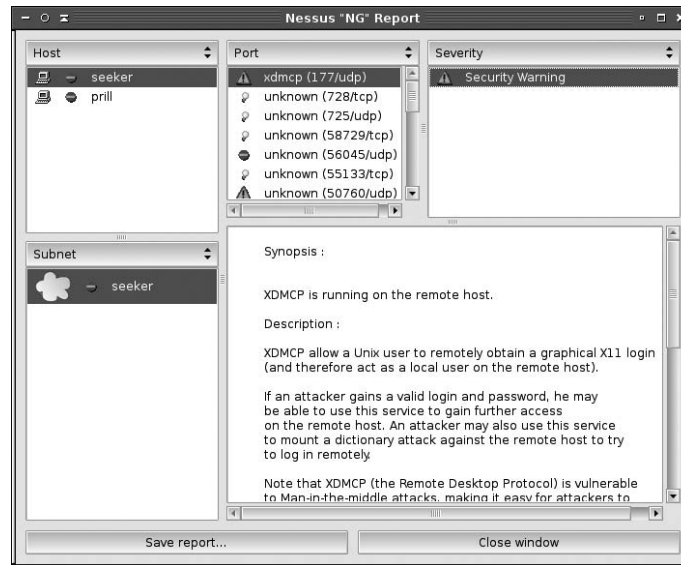


The Scan Options tab deserves mention because you can set a port range to scan on this tab. Most servers run on the lowest 1024 ports, so scanning from 1–1024 is sufficient to detect most servers. A few servers run on higher ports, though. This is true of X (ports 6000–6099) and VNC (ports 5800–5999), to name a couple. A cracker might run a server on a nonstandard port, too, so if you want to detect abuse rather than just misconfiguration, you must scan all ports. The Target(s) field on the Target tab enables you to specify the computers you want to scan. You can list multiple targets by separating their hostnames by commas; Nessus will then scan them simultaneously.

Once you've set all the Nessus options, click Start the Scan. Nessus begins the port scan, which can take several minutes, or sometimes much longer, depending on the options

you've selected and the speed of the network connections between the Nessus server and the target systems. Nessus presents a summary screen as the scan is in progress. When the scan completes, Nessus presents its report, as shown in Figure 12.2. You can browse each scanned computer's open ports to learn what risks are associated with each one. You can look up many servers in this book or in other documentation to learn how to disable them, if you decide to do so.

**FIGURE 12.2** The Nessus scan report summarizes open ports and describes the nature and severity of the risks associated with each open port.



When you use a network scanner, you should consider the fact that the ports you see from your test system may not be the same as those that might be visible to an attacker. This issue is particularly important if you're testing a system that resides behind a firewall from another system that's behind the same firewall. Your test system is likely to reveal accessible ports that would not be accessible from the outside world. On the other hand, a cracker on your local network would most likely have access similar to your own, so you shouldn't be complacent because you use a firewall. Nonetheless, firewalls can be important tools for hiding servers without shutting them down.

## Reviewing Accounts

Your computer's accounts are a potential source of vulnerability. If accounts go unused but remain active, an intruder could conceivably obtain a password and break in. Even system accounts (those used by Linux itself to run servers or for purposes other than managing ordinary users) can pose a threat. An unused system account could be converted into a login account and used by an intruder, possibly escaping notice.

From time to time, you should study your local accounts by perusing the `/etc/passwd` and `/etc/shadow` files. These files and their contents are described in more detail in Chapter 5. Pay particular attention to the following security issues:

**Unknown accounts** Most Linux systems have many system accounts, and you might not remember them all, so don't jump to conclusions, but if you see an account you don't recognize, you should investigate it. Check its characteristics for any suspicious features, compare your current file with a backup made after system installation, and check log files for activity involving the account.

**Accounts with a UID of 0** Linux uses a UID of 0 to represent root, so any account with a UID of 0 other than root is *highly* suspicious. Attackers sometimes create such accounts or change the UID of an existing account to 0 in order to give themselves root privileges on a system.

**System accounts with passwords** System accounts, such as `daemon` and `cron`, are frequently used by Linux to run servers or other tools without root privileges. Ordinarily, these accounts don't need passwords, so if you see a password for such an account in `/etc/shadow`, it's very likely to be an indication of an intrusion. If you use shadow passwords, all accounts have an `x` in the `/etc/passwd` file's password field, so you must check the `/etc/shadow` file. Accounts without passwords are indicated by an exclamation mark or an asterisk in the password field, which is the second field in this file.

**Login shells** The login shell is the final field in the `/etc/passwd` file. Most system accounts use `/bin/false` or `/dev/null` as a login shell, although there are a few exceptions. Most notably, the shutdown account uses `/sbin/shutdown`, the `halt` account uses `/sbin/halt`, and root uses a normal shell (typically `/bin/bash`). A few server packages create accounts with normal shells as login programs, too, although most don't. This practice varies with the server program and distribution. As a general rule, though, you should be suspicious of system accounts with login shells other than `/bin/false` or `/dev/null`.

One of the best ways to review your accounts is to keep backups of `/etc/passwd` and `/etc/shadow` on a write-protected removable medium, such as a write-protected floppy disk. You can then mount that disk and compare the backup to your on-disk file using `diff`:

```
diff /etc/passwd /mnt/floppy/passwd
diff /etc/shadow /mnt/floppy/shadow
```

If you haven't added, deleted, or modified accounts, these commands should return no output lines. If accounts have been changed, `diff` will summarize the changes. Note that changes to `/etc/shadow` include password changes, so this comparison is likely to turn up many changes on a multiuser system, particularly if users are diligent about changing their passwords on a regular basis.



Password files stored on floppy disks pose a security threat themselves. They should be kept under lock and key—ideally in a safe that can be accessed only by system administrators who can ordinarily read the original files.

## Verifying Installed Files and Packages

A final method of security auditing is verifying installed files and packages. One approach to doing this is to use a package tool such as RPM, as described later in “Using Package Manager Checksums.” This procedure will help look for sloppily replaced program files. It won’t help you spot changes to files in programs you didn’t install via your package manager, though. It’s also overly sensitive to changes to configuration files, which you often alter yourself after installing the package. Tripwire (described later in “Using Tripwire”) is another tool that can be used in this way. It’s more helpful in spotting changes to key configuration files, but it’s more of a hassle to use.

Another approach you can take is to keep backups of known-good configuration files on read-only media. You can then compare your current configuration files to the backups from time to time. Using `diff`, as described in “Reviewing Accounts,” can be an effective way to do this.

## Intrusion Detection

Even the best-configured computer has vulnerabilities. With luck, these vulnerabilities won’t be exploited, but you shouldn’t make that assumption. Instead, you should actively search for evidence of intrusions on your systems. That way, you’ll at least be alerted to an intrusion and be able to take appropriate steps soon after the intrusion occurs. “Appropriate steps” are usually a complete reinstallation or restore from a clean backup, followed by tightening security around suspected points of entry. Once a system has been rooted, you can’t completely trust *anything* on that system, and restoring everything from a known-clean source is usually easier than checking each and every file for signs of tampering.

Several methods of detecting intruders exist. These range from being alert to suspicious activities to using assorted programs that can monitor network activity or check for changes in critical files.

## Symptoms of Intrusion

One way of detecting intrusion is to notice abnormalities in your system’s operation. This approach is unreliable, but it’s also the most general approach and might therefore succeed even if the intruder has a way to defeat more specific monitoring tools you might employ. The basic approach sounds simple: know how your system normally behaves, and be alert to changes in this behavior. Symptoms of intrusion can include the following:

**System slowdown** Intruders might run programs on your computer that cause it to respond more slowly than usual to its normal workload.

**Increased network activity** Just as intruders can consume CPU time, they can consume network resources. In fact, one reason crackers break into computers is to use your network connectivity. They may be using your system to launch a DDoS attack against others, to distribute files on an illegal FTP server, or for other purposes that consume a lot of network bandwidth.

**Changed program behavior** Crackers often replace standard system tools with their own customized versions. These tools may behave slightly differently from the originals, so changes in the way common utilities behave (such as text that's output differently or a complaint that an option that worked yesterday is no longer working) can be a clue to a system compromise.

**System or software crashes** If the computer, or just individual programs, begin crashing for no apparent reason, it could be because they've been modified by an intruder. In some sense, this is just a corollary of the changed program behavior symptom, but it's more severe.

**Mysteriously altered data files** If ordinary data files are changed without your having changed them, one possible explanation is an intrusion. In fact, depending on the nature of the change, it may be a flashing neon sign. For instance, intruders sometimes break in with the exclusive goal of defacing a Web site, so such a change is a virtual guarantee of system compromise.

**Missing or corrupted log files** Intruders often try to cover their tracks by altering or deleting log files. (Chapter 4 covers log files in more detail.) Therefore, such changes should raise a red flag.

**Off-site complaints** If the administrator of another site contacts you and complains of attacks from your site or other suspicious behavior, it may be that your computer has been compromised and is being used to attack another system.

**Local user complaints** Local users can be alert to certain types of problems, such as system sluggishness and program behavior changes. Listen to their complaints, and investigate them promptly.

The problem with using any of these symptoms is that they can all have causes other than an intrusion. A local program might be running out of control, consuming CPU time or network bandwidth; programs can behave differently or crash because of legitimate program upgrades, emerging hardware defects, or changed environment settings; data files and log files can be altered because of legitimate activities of other users or disk errors; off-site complaints can be generated in error or might be traced to rogue local users who have *not* gained inappropriate local access; and local user complaints can reflect any of these or many other nonsecurity problems. Nonetheless, being alert to these clues can lead you to investigate the problem and fix it, whether it turns out to be a security problem or not.

## Using Snort

Snort (<http://www.snort.org>) is a powerful *packet sniffer* program—a program that monitors packets directed to its host computer (or to other computers on its local network segment). Packet sniffers are very powerful network diagnostic tools because they enable you to dig into the “guts” of a network transaction. The knowledge you can gain from such investigations can help you diagnose problems of all sorts, but it requires extensive knowledge of the underlying protocols.



Packet sniffers are also popular tools among crackers. Packet sniffers can help intruders discover users' passwords and other sensitive data. In fact, using a packet sniffer can be grounds for disciplinary actions in many organizations. Thus, although Snort is a very good and legitimate tool when used properly, you shouldn't install and use Snort unless you have the authorization to do so.

In addition to functioning as a generic packet sniffer, Snort can function in a more sophisticated role as an *intrusion detection system (IDS)*. An IDS is a program that monitors network traffic for suspicious activity and alerts an appropriate administrator to warning signs. Put another way, an IDS is a packet sniffer that can recognize activity patterns that indicate an attack is underway.

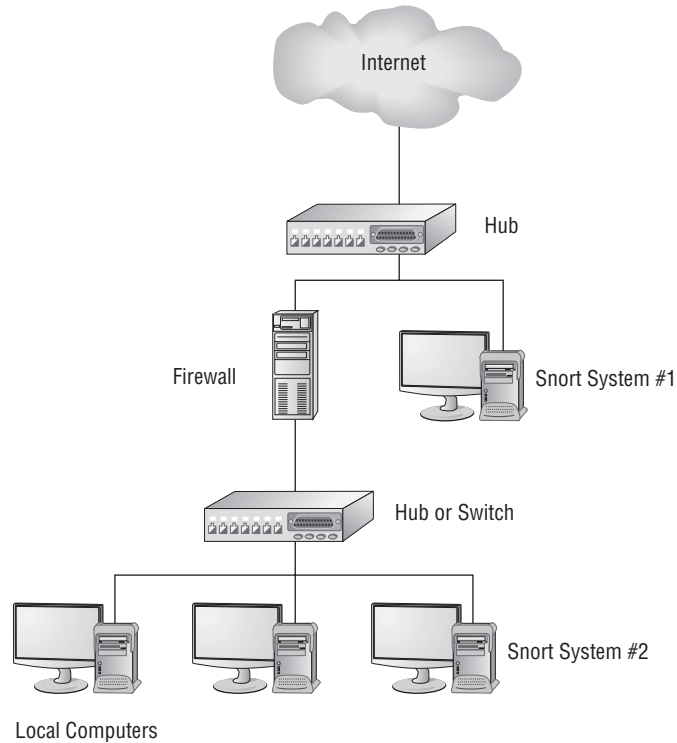
The first step to installing Snort is deciding where to place it. Figure 12.3 shows a couple of possible locations. Snort System #1 in this figure can monitor traffic to or from the Internet at large, while Snort System #2 can monitor local traffic. Both have a chance of catching outside attacks against specific local computers, but System #1 will be sensitive to attacks that are blocked by the firewall, while System #2 will be sensitive to purely local attacks. Also, System #2 requires either a hub rather than a switch locally or a switch that's programmed to echo all traffic to Snort System #2; a switch without such configuration will hide most traffic between the local computers from the Snort system, rendering it useless.

Most modern Linux distributions ship with Snort, so you should be able to install it in the usual way. Once installed, Snort is configured through the `/etc/snort/snort.conf` file. (Some distributions don't provide a file of this name but do provide a file called `snort.conf.distrib` or some other variant. You can copy or rename this file and use it as a template that you can modify.) A default `snort.conf` file may work acceptably, but you may want to customize several variables, such as `$HOME_NET`, `$SMTP_SERVERS`, and `$HTTP_SERVERS`. The first of these specifies the IP addresses to be monitored. Others define the IP addresses of particular types of servers. The default values tell Snort to monitor all IP addresses, which may be fine, since you may want Snort to watch all traffic on its local network, which is all it will ordinarily be able to see.

Some distributions place a series of supplementary Snort configuration files, with names that end in `.rules`, in the `/etc/snort` directory. These rule files define the sorts of packets that Snort should consider suspicious. Most protocols have a single `.rules` file, such as `smtp.rules` for SMTP packets. These `.rules` files are referenced via `include` directives in the main `snort.conf` file, so be sure your main `snort.conf` file loads the appropriate rules for your network. If you don't see a `.rules` file for a protocol you want to monitor, check <http://www.snort.org/snort-rules/>. This site hosts many Snort `.rules` files for less popular protocols.

To launch Snort, type its command name: **snort**. The program runs and logs its output in files located in `/var/log/snort`. These log files record information on suspicious packets. You should be sure to monitor these log files on a regular basis. To launch Snort on a permanent basis, you can run it from a startup script. In fact, many distributions provide SysV startup scripts to launch Snort.

**FIGURE 12.3** A Snort system can be placed at any of several locations to monitor network activity.



Snort doesn't need an IP address to monitor network traffic. Thus, you can configure a dedicated Snort system with network drivers but without an IP address and use it to monitor network traffic. This configuration makes the Snort monitor very resistant to external attacks, because an attacker can't directly address the system. On the downside, you must use the Snort system's own console or an RS-232 serial link to it to monitor its activities.

## Using PortSentry

Another IDS is PortSentry (<http://sourceforge.net/projects/sentrytools/>). The basic idea behind PortSentry is similar to that of Snort, in the sense that both are designed to alert you to suspicious network activity. One critical difference is that PortSentry runs on individual computers to monitor access attempts to their own ports, whereas Snort can monitor an entire network. Another difference is that PortSentry can actively block network scans. In fact, in some sense PortSentry is more like a firewall than an IDS.



After installing PortSentry (usually from a package called `portsentry`), you can configure it via its `portsentry.conf` file, which is generally found in `/etc` or `/etc/portsentry`. You specify ports you want PortSentry to monitor with the `TCP_PORTS` and `UDP_PORTS` options, which both specify comma-delimited lists of ports. PortSentry binds to these ports, logs attempts to access them, and can take various optional actions based on additional options in the PortSentry configuration file. These actions can include ignoring the access attempts, running external programs, dropping routes from the routing table, and so on.

## Using Wireshark

Wireshark (<http://www.wireshark.org>), formerly known as Ethereal, is another packet sniffer. Where Snort and PortSentry are popular IDSs, though, Wireshark is more often used as a network diagnostic tool. It features deep analysis of network packets, meaning that the program can help you dig into network packets to discover the meanings they carry, thus enabling you to track down network problems—for instance, is a failure to authenticate due to a mismatched protocol version, a mistyped password, packets that are going missing, or some other problem?



Wireshark, just like Snort, can be abused. Before using it, obtain written permission from your superiors, lest you run afoul of policies forbidding the unauthorized use of packet sniffers.

The most basic use of Wireshark is via the `tshark` command, which by default displays raw packet data intercepted by the program:

# **tshark**

Running as user "root" and group "root". This could be dangerous.

Capturing on eth1

```
0.000000 08:10:74:24:1b:d4 -> Broadcast ARP Who has 192.168.1.1? Tell
192.168.1.254
```

```
0.308644 192.168.1.8 -> 192.168.1.2 SSH Encrypted response packet len=128
```

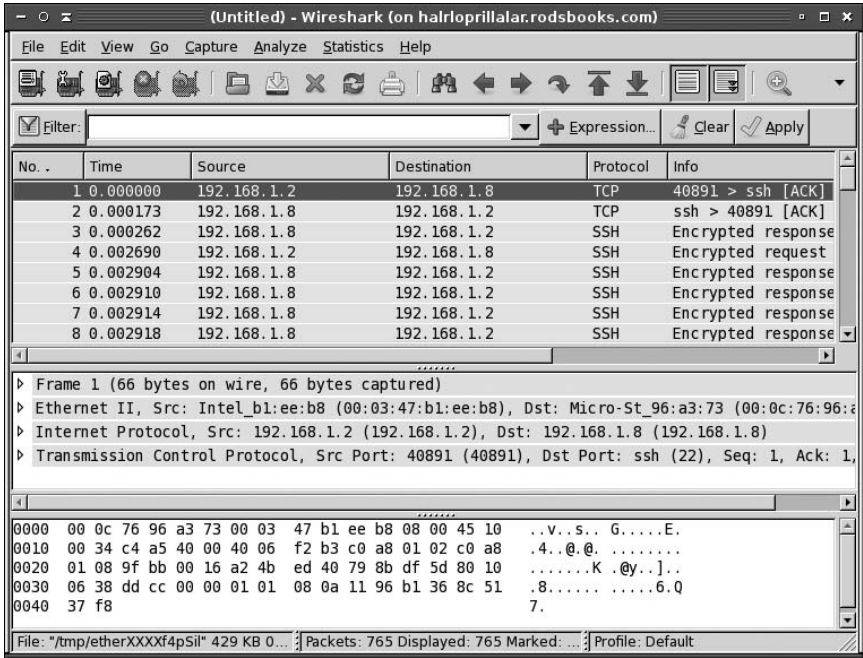
```
0.308721 192.168.1.8 -> 192.168.1.2 SSH Encrypted response packet len=48
```

This example (which is truncated; real output will continue indefinitely) shows an Address Resolution Protocol (ARP) broadcast followed by two SSH packets. If you want to restrict Wireshark's output, you can do so with a very wide array of `tshark` options. Consult its man page for details.

You'll probably find it easier to use Wireshark via its GUI interface, which is often installed via a separate package from the main Wireshark package. On Fedora, for instance, you must install `wireshark-gnome`. You can then type **wireshark** to launch the main Wireshark GUI program. Clicking Capture ➤ Start then begins a capture of all traffic. (You can limit what's captured by using the Capture ➤ Capture Filters option.) Clicking Capture ➤ Stop terminates the capture operation. The result resembles Figure 12.4. This window has three panes. The

top pane presents a summary of each network packet, the middle pane presents protocol-specific analysis of the packet selected in the top pane, and the bottom pane shows the raw packet data.

**FIGURE 12.4** Wireshark provides a GUI interface that enables point-and-click study of captured network traffic.



## Using Tripwire

Monitoring network traffic is a useful strategy for detecting undesirable activity, but it isn't guaranteed to detect an intruder. Perhaps the cracker is using an authorized protocol from an authorized location, for instance, and so the intrusion attempt doesn't trip any triggers in Snort or PortSentry. Should somebody manage to break into your computer, Tripwire (<http://www.tripwire.org>) may be your best bet to detect that fact. This utility records a set of information about all the important files on a computer, including various types of *checksums* and *hashes*—short digital “signatures” that enable you to quickly determine whether a file has been changed. (These can also be used in other ways; for instance, Linux uses hashes to store passwords.) With this database stored in a secure location, you can check your system periodically for alteration. If an intruder has modified any of your files, Tripwire will alert you to this fact. If you like, you can run a Tripwire verification on a regular basis—say, once a week in a cron job.

Many distributions ship with Tripwire, but it may not be installed by default. The utility is controlled through two configuration files: `tw.cfg` and `tw.po1`, which often reside in `/etc/tripwire`. The `tw.cfg` file controls overall configuration options, such as where `tw.po1` resides, how Tripwire sends reports to the system administrator, and so on. The `tw.po1` file includes information on the files it's to include in its database, among other things. Both files are binary files created from text-mode files called `twcfg.txt` and `twpo1.txt`, respectively. You may need to edit `twpo1.txt` to eliminate references to files you don't have on your system and to add information on files you do have but that the default file doesn't reference. Use the `twinstall.sh` program (which often resides in `/etc/tripwire`) to generate the binary configuration files and other critical database files. This utility will ask you to set a pair of passphrases, which are like passwords but are typically longer, to control access to the Tripwire utilities. You'll then need to enter these passphrases to have the utility do its encoding work.

Once you've generated the basic setup files, type **`tripwire --init`** to have it generate initial checksums and hashes on all the files it's configured to monitor. This process is likely to take a few minutes. Thereafter, typing **`tripwire --check`** will check the current state of the system against the database, and typing **`tripwire --update`** will update the database (say, in case you upgrade a package). The `--init` and `--update` operations require you to enter the passphrase, but `--check` doesn't. Therefore, you can include an automated Tripwire check in a cron job.



Tripwire is best installed and initialized on a completely fresh system, before connecting the computer to the Internet but after all programs have been configured. Although it's possible to install it on a system that's been up and running for some time, if that computer has already been compromised without your knowledge, Tripwire won't detect that fact.

## Generating Checksums Manually

If Tripwire is overkill, you can deploy a manual check for changes to files by using checksum programs, which generate short strings that are likely to be unique. Both `md5sum` and `sha1sum` create such checksums, although they generate different checksums:

```
$ md5sum /etc/passwd
f7ba3e97209ca5c4c8040276db5dc329 /etc/passwd
$ sha1sum /etc/passwd
3fc1d2e5d4ec5bf09d12beaf5a613cbf7f7da5f1 /etc/passwd
```

If you keep checksums of critical files, you can easily spot changes to them—if the checksum value has changed, then the file has changed. The key to such comparisons is to take a set of checksums before the computer could have been compromised; to do any good, a checksum must have a comparison value. If the file might change legitimately, such as `/etc/passwd`, remember to take a new checksum after you make any legitimate changes.



Checksum programs are often used on the Internet at large to help users spot corrupt file downloads. A program author may provide a tarball, RPM, or CD-R image file along with an MD5 sum. You can then check the file you download to be sure its MD5 sum matches what the author has posted. If it doesn't match, you can check your download settings (to be sure you're using FTP binary mode, for example), try again and, if the problem persists, report it to the program's author.

## Using Package Manager Checksums

Package managers—most notably the RPM Package Manager (RPM)—maintain checksums on all their installed packages. As such, they can be used as intrusion detection tools. In particular, the `-V` (or `--verify`) option to `rpm` performs a package verification:

```
rpm -V postfix
S.5....T c /etc/postfix/main.cf
S.5....T c /etc/postfix/sasl_passwd
S.5....T c /etc/postfix/sender_canonical
```

Each line of output reports files that have changed in some way. The first eight characters of the output lines report what has changed: the file size, mode, MD5 sum, device major or minor numbers (for device files), link path mismatch, user ownership, group ownership, or time. A dot (.) in a position indicates that a test passed; any other character denotes a change (the character used depends on the test and is intended to be mnemonic). Following the eight-character block may be another character that denotes the file type—`c` for configuration files, `d` for documentation, `g` for “ghost” files (those not included in the actual package), `l` for a license file, or `r` for a README file. Files that haven't changed are *not* displayed in the output.

In the preceding example, three files have changed: `main.cf`, `sasl_passwd`, and `sender_canonical`. All three files are marked as configuration files (the `c` characters preceding the filenames), and all three have changed file sizes, MD5 sums, and times. Because these are configuration files, these changes aren't particularly suspicious, but a similar pattern of changes in program executables would be cause for concern. Changes to the MD5 sum (the form of checksum used by RPM) are particularly likely to be the result of tampering; they indicate that the file's contents have changed compared to the file in the original package. Time stamp changes can sometimes be completely innocent. Ownership and permissions changes might be the result of unwanted tampering or could be innocent in some cases (say if you've deliberately removed world execute permission to improve security).

You can verify an individual package by providing a package name, as in the preceding example. You can also verify all the packages installed on a system by passing the `-a` option. The result is likely to be a very long list, though, because so many packages include configuration files and other ancillary files that are normally changed. You might want to pass the output to a file that you can peruse later, as in `rpm -Va > rpm-test.txt`.



One major limitation of a package manager's checksums for detecting intruders is that it's very easily overcome. Unlike Tripwire's database, the RPM database isn't password-protected. In fact, intruders can easily mask their tracks by using RPM itself to install their modified tools. When you attempt to use RPM to verify the package, RPM will merrily report no problems.

## Using *chkrootkit*

The *chkrootkit* program (<http://www.chkrootkit.org>) is something of a last-resort method of detecting intrusion and is the closest thing in the Linux world to Windows virus scanners. (Linux virus-scanning programs also exist, but they're intended mainly to check for Windows viruses on Samba shares. Linux viruses are not a problem in the real world, at least not as of mid-2009.)

Many crackers use *root kits*, which are prepackaged intrusion tools. When an intruder runs a root kit against a target, the root kit software probes for known weaknesses (such as servers with known security bugs), breaks in, and installs software to enable simpler access by the intruder. The intruder can then log in using Telnet, SSH, or the like, and gain full control of the system.



Intruders who use root kits are often referred to as *script kiddies*. These miscreants have minimal skill; they rely on the root kit to do the real work of the intrusion. Some people prefer to reserve the term "cracker" for more skilled intruders, but others consider script kiddies to be crackers with minimal expertise.

Using *chkrootkit* is fairly straightforward: type its name. The result is a series of lines summarizing checks that the software performs. These lines should all end with `not infected`, `no suspect files found`, or a similar reassuring message. If any message alerts you to an intrusion, you should take immediate corrective measures.

## Monitoring Log Files

Log files can provide clues to intrusions. Chapter 4 describes log files in detail, so you should consult it for basic information on where log files reside, how you can configure them, and how you can monitor them. You should be aware, though, that log files often contain clues about intrusions. Suspicious items that may appear in log files include:

**Suspicious logins** A login might catch your eye for any number of reasons. Perhaps the user is on vacation with no network access, or perhaps the login is from a location to which the user has no access. If you notice such activity, you should investigate further.

**Repeated login failures** Crackers sometimes attempt to log in by guessing passwords. This procedure is likely to leave a trace in log files in the form of a long series of login

failures. Just one failure followed by a successful login isn't very suspicious, though; this sort of pattern is more likely the result of a mistyped or momentarily forgotten password.

**Missing entries** Intruders often try to cover their tracks by deleting entries from log files. The result is suspicious gaps in the log files. For instance, if a system normally generates an average of one entry per minute, a gap of 20 minutes could signal that an intruder broke in and then deleted the log entries that would provide clues as to how this was done.

**Entries for servers that shouldn't be running** If a cracker launches a new server to facilitate future logins or perform some other task, you may see log entries from this new server.

Unfortunately, monitoring log files for all of these things can be tedious. Log file monitoring tools, such as those described in Chapter 4, can help minimize this tedium, but some discoveries are still likely to be serendipitous unless you spend all your time watching your log files grow.

## Summary

Linux's security mechanisms can help you keep your system from falling under the control of those who want to do you harm, or who simply want to abuse your system for their own ends. Like any OS's security measures, though, Linux's tools are only as good as their configurations. Although the default security of Linux systems has improved greatly since the late 1990s, maintaining a Linux system still requires that you understand the security tools available to you and that you be able to configure those tools to suit your needs.

Security begins with physical measures. With direct access to your computer, a miscreant can do anything at all, from changing configuration files to stealing your data to stealing the hardware. Many physical security measures are common sense, but others are very computer-specific.

Beyond physical security, firewalls and super server configurations can help protect the servers running on your computer. These options enable you to block access to your servers from undesired sources or to otherwise limit access to the computer.

Even the best configurations sometimes fail, so you should attend to the possibility by monitoring your system for intrusion. Various intrusion detection tools, such as Snort and Tripwire, will help you to do this. Don't neglect basic vigilance, though—if you notice something odd about how your system is behaving, that may be a sign of intrusion. You should also periodically review your security measures and look for weaknesses. Again, various tools, such as `netstat` and `nmap`, can help you in this task.

## Exam Essentials

**Identify some common symptoms of a compromised computer.** Intruders often make mistakes when invading a system. These mistakes can manifest themselves as a sluggish system, a system that suddenly consumes more network bandwidth than usual, programs that suddenly begin crashing, programs that don't behave as they normally do, or other strange changes in the system's operation.

**Explain the purpose of GPG.** GPG is a tool that encrypts or decrypts files or e-mail messages. It can also generate a digital signature so as to verify to recipients that an unencrypted message originated from the claimed sender.

**Describe the function of NIS.** The Network Information Service is a way to centralize Linux account, hostname, and other local system and network information on a single server. Using NIS enables you to maintain one account database rather than duplicate this information on many computers.

**Describe the differences between Snort and PortSentry.** Snort can monitor network activity directed at multiple computers, given appropriate network infrastructure, thus providing an early alert system for the network as a whole. PortSentry, by contrast, is a tool that's designed to monitor and restrict access to a single computer.

**Summarize the tools that can be used for locating open ports.** Local open ports can be found with the `netstat` program, which uses local system calls to locate ports that are currently open. The `nmap` and `Nessus` programs can locate open ports on the local computer or on other computers by sending network probes to all or a subset of the ports on the target computer.

**Explain how corrupted files may be located.** Several tools can locate corrupt files, typically by using checksums to determine whether files on disk have been changed. These tools include Tripwire and the RPM system. Manually performing such comparisons using `diff` and backup files can also be effective; or you can store hashes using `md5sum` and `sha1sum`.

## Review Questions

1. A large installation requires a dozen system administrators, each of whom manages a particular set of servers on several computers. Which of the following tools could help prevent accidental or malicious damage by one administrator to another's configurations?
  - A. Tripwire
  - B. SELinux
  - C. GPG
  - D. RADIUS
2. Which of the following methods can be used to completely disable SELinux? (Choose all that apply.)
  - A. Adding `selinux=0` to the kernel boot options in the system's boot loader
  - B. Changing the SELINUX item in `/etc/selinux/config` to `disabled`
  - C. Typing `echo 1 > /selinux/enforce`
  - D. Typing `selinux disable`
3. At what point during system installation should you configure Tripwire?
  - A. Prior to installing major servers like Apache
  - B. After installing major servers but before configuring them
  - C. After installing and configuring major servers but before connecting the computer to the Internet
  - D. After connecting the computer to the Internet and running it for one to four weeks
4. Which of the following are network packet sniffers? (Choose all that apply.)
  - A. Wireshark
  - B. LDAP
  - C. Snort
  - D. Tripwire
5. What tool might you use to detect unauthorized servers running on a subnet you manage?
  - A. PAM
  - B. Nessus
  - C. SELinux
  - D. GPG



6. Which of the following tools is best suited for monitoring activity directed at multiple computers?
- A. LILO
  - B. PortSentry
  - C. Snort
  - D. SWAT
7. Why might you configure a Linux computer to function as an NIS client?
- A. To mount remote filesystems as if they were local
  - B. To defer to a network's central authority concerning user authentication
  - C. To set the system's clock according to a central time server
  - D. To automatically obtain IP address and other basic network configuration information
8. Which of the following programs uses local system calls to locate local ports that are currently open?
- A. netstat
  - B. nmap
  - C. chkrootkit
  - D. nessus
9. The `/etc/nsswitch.conf` file on a computer includes the following lines. What can you say about this computer?
- ```
passwd: compat ldap
shadow: compat ldap
group:  compat ldap
```
- A. The system uses LDAP but not local files to authenticate user passwords.
 - B. The system uses local files and LDAP to authenticate user passwords.
 - C. The system uses local files and LDAP to identify valid accounts.
 - D. The system uses LDAP to encrypt the local password database.
10. When might you need to run `nmap` as `root`?
- A. When scanning computers on a remote network
 - B. When scanning TCP ports above 1024
 - C. When scanning TCP ports below 1024
 - D. When scanning UDP ports

11. A Nessus port scan reveals that ports 22 and 25 are open on a computer. What steps should you take?
 - A. Shut down the programs that are using those ports.
 - B. Activate a firewall program to block access to those ports.
 - C. Nothing; those ports should normally be open.
 - D. The correct action depends on the system in question.
12. You've downloaded a GPG public key from a Web site into the file `fredkey.pub`. What must you do with this key to use it?
 - A. Type `inspect-gpg fredkey.pub`.
 - B. Type `gpg --readkey fredkey.pub`.
 - C. Type `import-gpg fredkey.pub`.
 - D. Type `gpg --import fredkey.pub`.
13. You want to send an encrypted message to an e-mail correspondent. You both have GPG. What must you send or receive before you can send your encrypted message?
 - A. You must send your GPG public key to your correspondent.
 - B. You must send your GPG private key to your correspondent.
 - C. You must obtain your correspondent's GPG public key.
 - D. You must obtain your correspondent's GPG private key.
14. Which of the following programs can locate open ports on the local computer or on other computers by sending network probes to ports on target computers? (Choose all that apply.)
 - A. `netstat`
 - B. `nmap`
 - C. `chkrootkit`
 - D. `nessus`
15. Which of the following tools can create a number that you can store in a file to be used in verifying at a later date that another file has not changed? (Choose all that apply.)
 - A. `sha1sum`
 - B. `file`
 - C. `md5sum`
 - D. `chkdsk`
16. Your workplace employs more than 100 users, who can move from one workstation to another in an open environment, necessitating the need to support login of any user at any workstation. What tool might you use to simplify account administration in this environment? (Choose all that apply.)
 - A. Winbind
 - B. Two-factor authentication
 - C. LDAP
 - D. NIS

17. You're configuring a Linux system to use a new authentication tool. What PAM module stacks are you likely to need to modify to enable the `login` program to authenticate users using the new tool?
- A. `auth` and `session`
 - B. `auth` and `account`
 - C. `account` and `session`
 - D. `account` and `password`
18. Which of the following is the best definition of a checksum?
- A. An encrypted version of the data that can be unencrypted
 - B. A mathematical inverse of the ASCII value of all the characters added together
 - C. A 56-bit encryption key used to protect the contents of a packet
 - D. A digital signature that enables you to determine whether data has been changed
19. You suspect that the `/etc/passwd` file may have been altered over the course of the evening. You want to compare the current file with an offline backup made last night. Which utility should you use to compare the two versions of the file?
- A. `tar`
 - B. `cpio`
 - C. `diff`
 - D. `check`
20. Where does PAM store its configuration files?
- A. `/var/spool/pam.d/`
 - B. `/var/spool/pam/`
 - C. `/etc/pam/`
 - D. `/etc/pam.d/`

Answers to Review Questions

1. B. SELinux enables creation of multiple administrative accounts, and each administrator can modify only certain subsystems. Using this system, the multiple administrators in the question could each have limited administrative access to the computers in question, preventing the sorts of problems outlined. Tripwire can detect unauthorized changes and so might be able to detect problems quickly, but it won't do what's asked. GPG is a personal encryption tool that won't solve the problem posed. RADIUS is a network authentication protocol that won't solve the problem posed.
2. A, B. Options A and B both describe methods of disabling SELinux, although option B will work only on some distributions. Option C will *enable* SELinux, not disable it. Option D describes a fictitious command.
3. C. Tripwire records checksums and hashes of major files, including server executables and configuration files. Thus, these files should be in place and properly configured before you configure Tripwire. Once the system has been running on the Internet, there's a chance that it has been compromised; you should install Tripwire prior to connecting the computer to the Internet in order to reduce the risk that its database reflects an already-compromised system.
4. A, C. Wireshark and Snort are both network packet sniffers, although their features differ, with Wireshark being optimized as a protocol analyzer and Snort as an intrusion detection system (IDS). LDAP is a directory server, which can hold Linux account information or other data. Tripwire can detect unauthorized changes to a computer's configuration.
5. B. Nessus is a port scanner, which detects servers running on computers. Thus, it will do the requested job. PAM is Linux's authentication tool. SELinux is a set of Linux extensions designed to enhance security. GPG is a personal encryption and identification tool. None of these three tools will do the requested job.
6. C. Snort can monitor network activity directed at multiple computers, given appropriate network infrastructure, thus providing an early alert system for the network as a whole. LILO is a boot loader, while PortSentry is designed to monitor and restrict access to a single computer. SWAT is the Samba Web Administration Tool.
7. B. NIS functions as a means of distributing database information across a network, most notably including user authentication information. It's not used for file sharing, clock setting, or distributing basic TCP/IP configuration information.
8. A. Local open ports can be found with the `netstat` program, which uses local system calls to locate ports that are currently open. The `nmap` and `nessus` programs can locate open ports on the local computer or on other computers by sending network probes to all or a subset of the ports on the target computer. `chkrootkit` is something of a last-resort method of detecting intrusion and is the closest thing in the Linux world to Windows virus scanners.

9. C. The `/etc/nsswitch.conf` file's `passwd`, `shadow`, and `group` lines control how Linux identifies valid accounts and groups, as option C specifies. This file doesn't control actual authentication, as options A and B specify; that task is managed by PAM and its configuration files. LDAP doesn't encrypt the local password database, as option D specifies, and `/etc/nsswitch.conf` doesn't control this database's encryption.
10. D. The `nmap` program can scan TCP ports as any user, but it must be run as `root` to scan UDP ports, as option D specifies. The account you use to run the program doesn't impact its ability to scan local vs. remote computers. Likewise, this has no effect on the ability to scan ports above or below port number 1024.
11. D. Open ports are normal on many computers, such as systems that function as network servers. Port 22 is used by the Secure Shell (SSH) login server, and port 25 is used by Simple Mail Transfer Protocol (SMTP) mail servers. Both ports might legitimately be open on many computers; however, if the computer you were scanning was a desktop computer that shouldn't be running those servers, it might be appropriate to shut down the servers, or at least block access to them. Thus, option D is correct given the limited information presented in the question, although any of options A, B, or C might be appropriate once you learn more about the system with the open ports.
12. D. Option D provides the correct command to import `fredkey.pub` prior to use. The `inspect-gpg` and `import-gpg` commands of options A and C are fictitious, and there is no `--readkey` option to `gpg`, as option B suggests.
13. C. The usual method of sending encrypted messages with GPG entails the sender using the recipient's public key to encrypt the message. Thus, option C is correct. Option A would be correct if your correspondent needed to send you an encrypted message, but the question only specifies your sending the encrypted message. Options B and D both entail delivery of private keys, which is inadvisable at best, because private keys in the wrong hands permit the holder to impersonate the person who owns the keys.
14. B, D. The `nmap` and `nessus` programs can locate open ports on the local computer or on other computers by sending network probes to all or a subset of the ports on the target computer. Local open ports can be found with the `netstat` program. `chkrootkit` detects intrusions on a Linux system.
15. A, C. The `sha1sum` and `md5sum` programs both compute checksums (based on the SHA1 and MD5 algorithms, respectively), which are what the question is asking for. The `file` program can identify a file type, but it won't generate a checksum. `chkdisk` isn't a standard Linux utility, although a DOS/Windows program of that name verifies a disk's overall integrity, similar to `fsck` in Linux.
16. A, C, D. Winbind enables Linux to authenticate against a Windows domain controller, so it can help in the specified scenario if you already have a Windows domain configuration. LDAP and NIS provide similar features, but with superior support for Linux-specific account information. Thus, A, C, and D are all correct responses. Two-factor authentication is a method for improving security by requiring two types of authentication, such as a username/password pair and a biometric test (a fingerprint match, for instance). Two-factor authentication won't help in the specified scenario; in fact, it's likely to complicate matters.

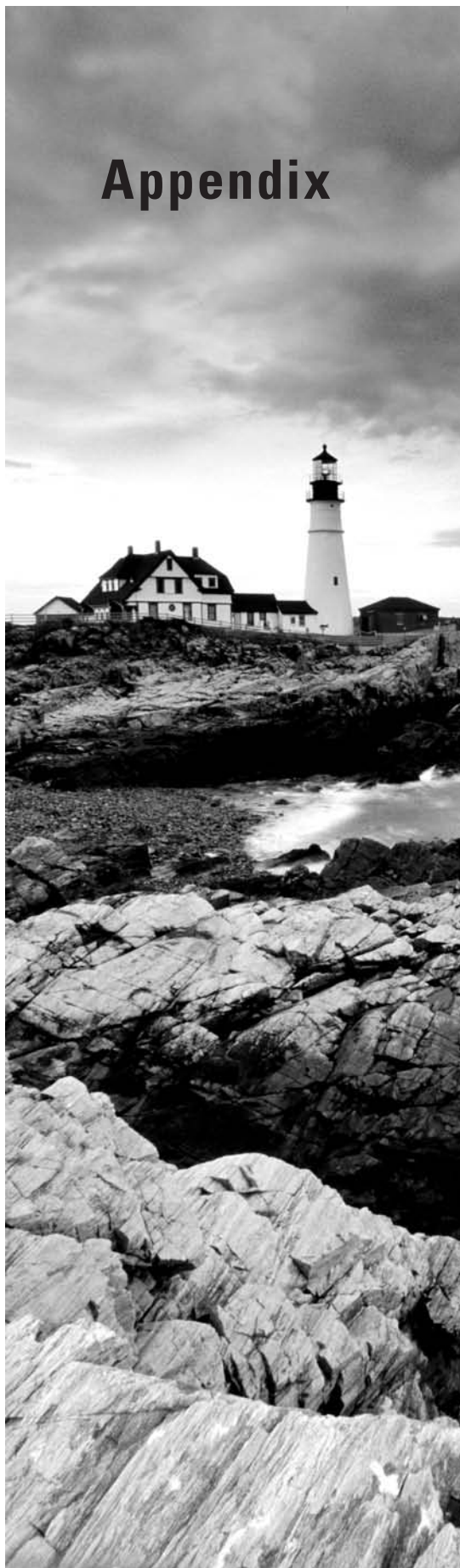
17. B. The `auth` and `account` stacks are the ones that are most involved in actual authentication and so should be changed for a login service such as the `login` program. The `session` stack manages login and logout bookkeeping tasks, and `password` manages password maintenance; these stacks are unlikely to require changing to use the new authentication tool.
18. D. A checksum is a short digital “signature” that enables you to quickly determine whether data has been changed. A checksum cannot be unencrypted, contrary to option A. Option B is a fictitious description. Option C is far too specific, and it doesn’t describe a checksum in any event.
19. C. The `diff` utility can be used to check two files for differences. Both `tar` and `cpio` are used for backing up files, but they will not compare them, and `check` is not a standard utility.
20. D. The standard location for PAM configuration files on modern Linux systems is `/etc/pam.d`. Some very old systems used a single PAM configuration file, `/etc/pam.conf`, but options A, B, and C have never been used as standard PAM configuration file locations.

Appendix

About the Companion CD

✓ In this appendix:

- What you'll find on the CD
- System requirements
- Using the CD
- Troubleshooting





What You'll Find on the CD

The following sections are arranged by category and summarize the software and other goodies you'll find on the CD. If you need help with installing the items provided on the CD, refer to the installation instructions in the "Using the CD" section of this appendix.

The programs on the CD might fall into one of these categories:

Shareware programs are fully functional, free, trial versions of copyrighted programs. If you like particular programs, register with their authors for a nominal fee and receive licenses, enhanced versions, and technical support.

Freeware programs are free, copyrighted games, applications, and utilities. You can copy them to as many computers as you like—for free—but they offer no technical support.

GNU software is governed by its own license, which is included inside the folder of the GNU software. There are no restrictions on distribution of GNU software. See the GNU license at the root of the CD for more details.

Trial, demo, or evaluation versions of software are usually limited either by time or by functionality (such as not letting you save a project after you create it).

Sybex Test Engine

The CD contains the Sybex test engine, which includes the assessment test and chapter review questions in electronic format, as well as two bonus exams located only on the CD.

PDF of the Book

We have included an electronic version of the text in PDF form. You can view the electronic version of the book with Adobe Reader.

Adobe Reader

We've also included a copy of Adobe Reader so you can view PDF files that accompany the book's content. For more information on Adobe Reader or to check for a newer version, visit Adobe's Web site at <http://www.adobe.com/products/reader/>.

Electronic Flashcards

For PC and Pocket PC

These handy electronic flashcards are just what they sound like. One side contains a question or fill-in-the-blank question, and the other side shows the answer.

System Requirements

Make sure your computer meets the minimum system requirements shown in the following list. If your computer doesn't match up to most of these requirements, you may have problems using the software and files on the companion CD. For the latest and greatest information, please refer to the ReadMe file located at the root of the CD-ROM.

- A PC running Microsoft Windows 98, Windows 2000, Windows NT4 (with SP4 or later), Windows Me, Windows XP, or Windows Vista. If you intend to run on Linux, a Web browser with the latest Adobe Flash Player plug-in is needed to run the Sybex Test Engine. 1024x768 or greater display resolution and a high color (16-bit or greater) video card are also required.
- An Internet connection.
- A CD-ROM drive.



Please read the CD's ReadMe file for more detailed instructions on using this CD with Linux.

Using the CD

To install the items from the CD to your hard drive, follow these steps:

1. Insert the CD into your computer's CD-ROM drive. The license agreement appears.



Windows users: The interface won't launch if you have autorun disabled. In that case, click Start > Run (for Windows Vista, Start > All Programs > Accessories > Run). In the dialog box that appears, type D:\Start.exe. (Replace *D* with the proper letter if your CD drive uses a different letter. If you don't know the letter, see how your CD drive is listed under My Computer.) Click OK.

**NOTE**

Linux users: If a Web browser with the license agreement doesn't appear when you insert the CD-ROM into the drive, first ensure that the drive is mounted. Typing `mount /mnt/cdrom` or `mount /media/cdrom` will usually do this. You can then launch a Web browser and enter `file:///mnt/cdrom/Start.html` in the URL entry field. (You may need to change `/mnt/cdrom` to some other string if your CD-ROM mount point differs from this.) Your system must have Adobe Flash (<http://labs.adobe.com>) installed and configured to work with your Web browser to use the test engine and flashcards.

2. Read the license agreement, and then click the Accept button if you want to use the CD.

The CD interface appears. The interface allows you to access the content with just one or two clicks.

Troubleshooting

Wiley has attempted to provide programs that work on most computers with the minimum system requirements. Alas, your computer may differ, and some programs may not work properly for some reason.

The two likeliest problems are that you don't have enough memory (RAM) for the programs you want to use or you have other programs running that are affecting installation or running of a program. If you get an error message such as "Not enough memory" or "Setup cannot continue," try one or more of the following suggestions and then try using the software again:

Turn off any antivirus software running on your computer. Installation programs sometimes mimic virus activity and may make your computer incorrectly believe that it's being infected by a virus.

Close all running programs. The more programs you have running, the less memory is available to other programs. Installation programs typically update files and programs; so if you keep other programs running, installation may not work properly.

Have your local computer store add more RAM to your computer. This is, admittedly, a drastic and somewhat expensive step. However, adding more memory can really help the speed of your computer and allow more programs to run at the same time.

Customer Care

If you have trouble with the book's companion CD-ROM, please call Wiley Product Technical Support at (800) 762-2974. Outside the United States, call +1(317) 572-3994. You can also contact Wiley Product Technical Support at <http://sybex.custhelp.com>. John Wiley & Sons will provide technical support only for installation and other general quality-control items. For technical support on the applications themselves, consult the program's vendor or author.

To place additional orders or to request information about other Wiley products, please call (877) 762-2974.

Glossary



Numbers

1024-cylinder limit The x86 BIOS has traditionally been unable to read past the 1024th cylinder in a cylinder/head/sector (CHS) addressing scheme, which has limited the size of hard disks—first to 504MB (or about 528 million bytes, so some people refer to it as the 528MB limit) and then to just under 8GB. On a computer with an old BIOS, the 1024-cylinder limit prevents the system from booting a kernel from higher than this limit, although Linux itself uses addressing schemes that aren't bothered by this limit. BIOSs made since the late 1990s also include ways around the limit, if the software understands those mechanisms. See also *cylinder/head/sector (CHS) addressing*.

3DES See *Triple Data Encryption Standard (3DES)*.

802.11 A family of wireless network protocols (often referred to as *Wi-Fi*). Examples include 802.11a (54Mbps, but uncommon), 802.11b (11Mbps), 802.11g (54Mbps), and 802.11n (600Mbps, which is expected to be finalized in December 2009). Most wireless routers, network cards, and wireless-capable laptop computers sold in 2009 may use 802.11b, 802.11g, and often an early subset of 802.11n.

A

absolute directory name A directory name that begins with a slash (/), indicating that it's to be interpreted starting from the root (/) directory.

access control list (ACL) A security system that provides a list of usernames or groups and their permissions to access a resource. ACLs can supplement traditional Unix-style permissions on all common Linux filesystems, although using ACLs requires setting a compile-time option in the kernel.

account Stored information and a reserved directory that allows one individual to use a computer. The term is often used and thought of as if it were a distinct virtual component of a computer that a person can use, as in “Sam logged into his account” or “Miranda's account isn't working.”

ACL See *access control list (ACL)*.

ACPI See *Advanced Configuration and Power Interface (ACPI)*.

Address Resolution Protocol (ARP) A protocol used to learn a network hardware address based on an IPv4 address. See also *Neighbor Discovery Protocol (NDP)*.

Advanced Configuration and Power Interface (ACPI) A power management protocol. Linux provides ACPI support.

Advanced Power Management (APM) A power management protocol. Linux includes better APM support than ACPI support.

Advanced Technology Attachment (ATA) A type of interface for hard disks, CD-ROM drives, tape drives, and other mass storage devices. Also often referred to as *EIDE*. Documents from before 2003 (and many from after that date) use “ATA” or “EIDE” to refer to what is called “PATA” today. See also *Parallel ATA (PATA)* and *Serial ATA (SATA)*.

AMD64 See *x86-64*.

American Standard Code for Information Interchange (ASCII) A method of encoding letters, numbers, and punctuation as numbers. For instance, ASCII code 65 corresponds to the uppercase letter *A*. ASCII was designed for encoding English, so it requires awkward extensions when dealing with many non-English languages. See also *Unicode Transformation Format (UTF)*.

APM See *Advanced Power Management (APM)*.

AppleTalk A network protocol stack used by Apple with its Macintosh computers. AppleTalk is used primarily on local networks for file and printer sharing.

ARP See *Address Resolution Protocol (ARP)*.

ASCII See *American Standard Code for Information Interchange (ASCII)*.

ATA See *Advanced Technology Attachment (ATA)*.

B

Bash The Bourne Again Shell, an open source variant of the Bourne shell. Bash is the default shell on most Linux installations.

Basic Input/Output System (BIOS) A low-level software component included on a computer’s motherboard in read-only memory (ROM) form. The CPU runs BIOS code when it first starts up, and the BIOS is responsible for locating and booting an OS or OS loader.

baud rate A measure of data transmission speed, commonly used over serial lines, corresponding to the number of signal elements transmitted per second. This term is often used as a synonym for “bits per second,” but many modems encode more than one bit per signal element, so the two aren’t always synonymous.

Berkeley Internet Name Domain (BIND) A common Domain Name System (DNS) server for Linux.

binary 1. The base-2 numbering system. 2. A program or file that contains data other than plain text, such as graphics or program data. 3. The version of a program that the computer runs, as opposed to the source code version of the program.

binary package A file that contains a compiled and ready-to-run Linux program, including necessary configuration files, documentation, and other support files.

BIND See *Berkeley Internet Name Domain (BIND)*.

BIOS See *Basic Input/Output System (BIOS)*.

bit A binary digit (0 or 1).

blowfish An encryption algorithm used by several important Linux security tools, such as SSL and SSH.

boot loader A program that directs the boot process. The BIOS calls the boot loader, which loads the Linux kernel or redirects the boot process to another boot loader.

boot sector The first sector of a disk or partition. The boot sector for a bootable disk or partition includes boot loader code, although this code may be absent from nonbootable disks or partitions. See also *boot loader*.

broadband 1. High-speed (greater than 200Kbps) Internet connections delivered to homes and small businesses. 2. Networking technologies that support simultaneous transmission of data, voice, and video.

broadcast A type of network access in which one computer sends a message to many computers (typically all the computers on the sender's local network segment).

build number A number identifying minor changes made to a binary package by its maintainer, rather than changes implemented by the program's author, which are reflected in the version number.

bus A data transfer mechanism within the computer, such as the SCSI bus or the memory bus.

byte An 8-bit number, typically represented as falling between 0 and 255.

C

C library (libc) Standard programming routines used by many programs written in the C programming language. The most common Linux C library is also referred to as GNU libc (glibc).

cache memory A fast form of memory that's used to temporarily hold a subset of a larger but slower memory store. When properly implemented, caches can improve system performance. Hard disks include RAM as cache for data on disk, and computers can implement their own disk caches. Modern CPUs include a form of cache for RAM, and some motherboards include the same.

CD See *Compact Disc (CD)*.

central processing unit (CPU) The main chip on a computer, which handles the bulk of its computational tasks.

CGI See *Common Gateway Interface (CGI)*.

checksum A simple file integrity check in which the values of individual bits or bytes are summed up and compared to a stored value for a reference version of the file. More sophisticated file integrity checks that use hashes are sometimes referred to as checksums, although technically the two are different.

child process A relative term referring to a process that another one has created. For instance, when you launch a program from a shell, the program process is a child process of the shell process.

chipset One or more chips that implement the main features of a motherboard or add-in board for a computer. The chipset is *not* the CPU, though; the chipset provides more specialized functions, such as the ability to control a hard disk or produce a video display.

CHS addressing See *cylinder/head/sector (CHS) addressing*.

CHS mode See *cylinder/head/sector (CHS) mode*.

CHS translation See *cylinder/head/sector (CHS) translation*.

CIFS See *Common Internet Filesystem (CIFS)*.

CLI See *command-line interface (CLI)*.

client 1. A program that initiates data transfer requests using networking protocols. 2. A computer that runs one or more client programs.

command prompt One or more characters displayed by a shell or other program to indicate that you should type a command. Many Linux distributions use a dollar sign (\$) as a command prompt for ordinary users and a hash mark (#) as a command prompt for root.

command-line interface (CLI) A program that interacts with the user in text mode, accepting typed commands as input and displaying results textually. See also *shell*.

Common Gateway Interface (CGI) A method of enabling a Web server to run scripts or compiled programs locally in order to generate dynamic Web content.

Common Internet Filesystem (CIFS) Name for an updated version of the Server Message Block (SMB) file sharing protocols. CIFS is implemented in Linux via the Samba suite. It's often used to share files with Windows computers.

Common Unix Printing System (CUPS) The dominant printing software for Linux.

Compact Disc (CD) A recording medium for music or data. (Data discs are usually called "CD-ROMs.") CDs may be created in mass quantities or individually; in the latter case, the discs are generally called CD-recordable (CD-R) or CD-read/write (CD-RW) discs.

compiler A program that converts human-readable source code for a program into a binary format that the computer runs.

Complementary Metal Oxide Semiconductor (CMOS) setup utility A part of the BIOS that gives the user the ability to control key chipset features, such as enabling or disabling built-in ports.

conditional expression A construct of computer programming and scripting languages used to express a condition, such as the equality of two variables or the presence of a file on a disk. Conditional expressions enable a program or script to take one action in one case and another action in the other case.

console 1. The monitor and keyboard attached directly to the computer. 2. Any command prompt, such as an `xterm` window.

Coordinated Universal Time (UTC) See *Greenwich Mean Time (GMT)*.

CPU See *central processing unit (CPU)*.

cracker An individual who breaks into computers. Crackers may do this out of curiosity, malice, for profit, or for other reasons.

creating a filesystem Writing low-level filesystem (meaning 1) data structures to a disk. This is sometimes also called high-level formatting. See also *filesystem*.

cron job A program or script that's run at a regular interval by the cron daemon. See also *system cron job* and *user cron job*.

CUPS See *Common Unix Printing System (CUPS)*.

cylinder/head/sector (CHS) addressing A method of hard disk addressing in which a triplet of numbers (a cylinder, a head, and a sector) are used to identify a specific sector. CHS addressing contrasts with linear block addressing (LBA).

cylinder/head/sector (CHS) mode See *cylinder/head/sector (CHS) addressing*.

cylinder/head/sector (CHS) translation Modifying one CHS addressing scheme into another. CHS translation was commonly used by BIOSs in the mid-to-late 1990s to enable the systems to use hard disks between 504MB and 8GB in capacity.

D

daemon A program that runs constantly, providing background services. Linux servers are typically implemented as daemons, although there are also nonserver daemons.

Data Display Channel (DDC) A protocol that enables a computer to query a monitor for its maximum horizontal and vertical refresh rates and other vital statistics.

DDC See *Data Display Channel (DDC)*.

DDoS attack See *distributed denial of service (DDoS) attack*.

Debian package A package file format that originated with the Debian distribution but is now used on several other distributions. Debian packages feature excellent dependency tracking and easy installation and removal procedures.

default route The route that network packets take if a more specific route doesn't direct them in some other way. The default route typically involves a gateway or router system that can further redirect the packets.

denial of service (DoS) attack A type of attack on a computer or network that prevents the use of a computer for its intended function, typically without actually breaking into the computer. These attacks frequently involve flooding a network or computer with useless data packets that overload the target's network bandwidth. See also *distributed denial of service (DDoS) attack*.

dependency A requirement of one software package that another one be installed. For instance, most Linux programs include a dependency on the C library.

desktop computer A computer that sits on a desk and that's used by an individual for productivity tasks. A desktop computer is similar to a workstation, but some people use "desktop" to refer to somewhat lower-powered computers or those without network connections. See also *workstation*.

desktop environment A set of programs that provide a friendly, graphical environment for a Linux user.

DHCP See *Dynamic Host Configuration Protocol (DHCP)*.

DHCP lease A temporary assignment of an IP address to a DHCP client by a DHCP server. Clients must periodically renew their DHCP leases or risk losing the right to use their IP addresses.

Digital Versatile Disc (DVD) A data storage medium similar to CDs, but with greater capacity (4.7GB–8.5GB). Like CDs, DVDs may be manufactured in quantity or recorded on a one-off basis; the latter include DVD-Rs, DVD+Rs, DVD-RWs, DVD+RWs, and DVD-RAMs.

direct memory access (DMA) A means of transferring data between devices (such as sound cards or SCSI host adapters) and memory without directly involving the CPU.

Disk Operating System (DOS) An early 16-bit x86 operating system. This OS is the basis for Windows 9x/Me, but not for Windows NT/200x/XP/Vista. DOS is sometimes used as a platform for disk partitioning tools or as a way to boot a Linux kernel.

disk quota A limit on the amount of disk space that an individual or group may use.

distributed denial of service (DDoS) attack A type of DoS attack in which the attacker uses many hijacked computers to cripple a computer with much better network connectivity than any one of the hijacked computers.

distribution A complete collection of a Linux kernel and programs necessary to do work with Linux. Dozens of different Linux distributions exist, each with its own unique characteristics, but they all work in a similar way and can run the same programs, assuming similar vintages of critical support libraries like libc.

DMA See *direct memory access (DMA)*.

DNS See *Domain Name System (DNS)*.

domain A collection of related computers. See also *domain name*.

domain name A name associated with an organization or set of computers. Individual computers are assigned names within a domain, and domains can be partitioned into subdomains.

Domain Name System (DNS) A distributed set of computers that run servers to convert between computer names (such as `ns.example.com`) and IP addresses (such as `192.168.45.204`). DNS servers are organized hierarchically and refer requests to systems responsible for successively more specific domains.

DOS See *Disk Operating System (DOS)*.

DoS attack See *denial of service (DoS) attack*.

dot file A Linux or Unix file whose name begins with a dot (.). Most Linux shells and programs hide such files from the user, so user configuration files usually come in this form to be unobtrusive in directory listings.

dpkg 1. An abbreviation for *Debian package*. 2. A program of the same name, used to manipulate Debian packages.

DRAM See *dynamic RAM (DRAM)*.

DVD See *Digital Versatile Disc (DVD)*.

Dynamic Host Configuration Protocol (DHCP) A protocol used on local networks for dissemination of network configuration information. A single DHCP server can maintain information for many DHCP clients, reducing overall configuration effort.

dynamic RAM (DRAM) One of several types of RAM. Plain DRAM is now largely obsolete in desktop computers.

E

effective user ID The owner associated with a running process. This may or may not be the same as the user ID of the individual who ran the program.

EFI See *Extensible Firmware Interface (EFI)*.

EIDE See *Enhanced Integrated Device Electronics (EIDE)* and *Advanced Technology Attachment (ATA)*.

EM64T See *x86-64*.

Enhanced Integrated Device Electronics (EIDE) Another name for the *Advanced Technology Attachment (ATA)* interface.

envelope In networking, the portion of a data packet that directs the transmission and routing of the packet. The envelope includes such information as the source and destination addresses and other housekeeping information.

environment variable A setting that's available to any program running in a session. Environment variables can define features such as the terminal type being used, the path to search for executable programs, and the location of an X server for GUI programs.

Ethernet The most common form of wired local networking.

ext2 See *Second Extended Filesystem (ext2 or ext2fs)*.

ext2fs See *Second Extended Filesystem (ext2 or ext2fs)*.

ext3 See *Third Extended Filesystem (ext3 or ext3fs)*.

ext3fs See *Third Extended Filesystem (ext3 or ext3fs)*.

ext4 See *Fourth Extended Filesystem (ext4 or ext4fs)*.

ext4fs See *Fourth Extended Filesystem (ext4 or ext4fs)*.

extended INT13 BIOS routines added in the late 1990s to enable x86 computers to boot from hard disks larger than 8GB.

extended partition A type of disk partition used on disks partitioned using the MBR partitioning scheme. Extended partitions are placeholders for one or more logical partitions. Extended partitions are not needed with GPT or most other non-MBR partitioning systems.

Extensible Firmware Interface (EFI) A system intended to replace the aging x86 BIOS. EFI provides numerous enhancements intended to help computers cope with modern hardware. The GPT partitioning system is part of the EFI specification, although GPT may be used on non-EFI systems as well.

Extent Filesystem (XFS) One of several journaling filesystems for Linux. XFS was developed by Silicon Graphics (SGI) for its IRIX OS and then ported to Linux.

external transfer rate The data transfer rate between one device and another. The external transfer rate is frequently applied to disks and similar devices in reference to the speed of the ATA or SCSI interface, as opposed to the speed of the drive mechanism itself. In this context, the external transfer rate is almost always higher than the internal transfer rate.

F

failed dependency A state in which a package's dependencies are not met when attempting to install it, or in which removing a package would cause other installed packages to have unmet dependencies.

FAT See *File Allocation Table (FAT)*.

FDDI See *Fiber Distributed Data Interface (FDDI)*.

Fiber Distributed Data Interface (FDDI) A type of network hardware that supports up to 100Mbps speeds over fiber-optic cables.

Fibre Channel A type of network hardware that supports up to 6800Mbps speeds over fiber-optic cables.

file access permissions Linux's file access control mechanism. Every file has an owner, a group, and permissions that define how the owner, group members, and all other users (the "world") may access the file. Permissions include read, write, and execute for the owner, group, and world.

File Allocation Table (FAT) 1. A filesystem (meaning 1) popularized by DOS and used today on removable media because of its wide support. See also *Virtual FAT (VFAT)*.
2. A data structure used on the FAT filesystem, after which the filesystem is named.

file owner The account with which a file is most strongly associated. The owner often has permission to do more with a file than other users can do.

file permissions See *file access permissions*.

file sharing protocol A network protocol that enables one computer to access files stored on a second computer as if the second computer's files were local to the first computer. Examples include SMB/CIFS (used on Windows-dominated networks), NFS (used on Unix-dominated networks), and AppleShare (used on Macintosh-dominated networks).

File Transfer Protocol (FTP) A common network protocol designed for the transfer of files between computers. In its basic form, FTP does not support encryption, so it's best reserved for use on well-protected local networks or to perform *anonymous* FTP access, in which publicly accessible files may be retrieved using no or any password.

file type code A special code that identifies the type of a file, such as a regular file, a directory, or a device file.

filename completion A feature of some shells that enables them to complete a command or filename when you press the Tab key.

filesystem 1. The low-level data structures recorded on a disk in order to direct the placement of file data. The filesystem determines characteristics such as the maximum partition size, the file-naming rules, and what extra data (time stamps, ownership, and so on) may be

associated with a file. 2. The overall layout of files and directories on a computer. For instance, a Linux filesystem includes a root directory (/), several directories falling off this (/usr, /var, /boot, etc.), subdirectories of these, and so on.

firewall 1. A program or kernel configuration that blocks access to specific ports or network programs on a computer. 2. A computer that's configured as a router and that includes firewall software that can restrict access between the networks it manages.

FireWire Apple's preferred name for IEEE-1394.

font server A program that provides font bitmaps to client programs on the same or (sometimes) other computers. The font server may work directly from font bitmaps, or it may generate the bitmaps from outline fonts such as PostScript Type 1 or TrueType fonts.

fork 1. The method by which one process creates another process. 2. The creation of a new software development project from an earlier one. Each project is then developed independently. For instance, X.org-X11 is a fork of XFree86.

forwarding-only DNS server A DNS server that doesn't perform a *full recursive DNS lookup* for clients but instead forwards the whole request to another DNS server. This configuration is common on small networks and can improve overall DNS performance for a network.

Fourth Extended Filesystem (ext4 or ext4fs) A variant of the *Third Extended Filesystem* (*ext3* or *ext3fs*) that adds the ability to use larger disks (up to 1 exabyte vs. 16 terabytes for *ext3fs*) and an assortment of performance enhancements.

fragmented Adjective describing files whose contents are split across several parts of a disk, rather than placed contiguously. File fragmentation tends to degrade disk performance because it increases head movements when reading files.

frame In networking, a data packet associated with network hardware (such as Ethernet), as opposed to the software (such as TCP/IP).

frame buffer A low-level but standardized interface between software and video hardware. X uses a frame buffer interface on many non-x86 computers.

Free Software Foundation (FSF) An organization, headquartered in Cambridge Massachusetts, which sponsors the development of open source software and advocates its use. The FSF created the General Public License (GPL) and the GNU project, which developed much of the software used in Linux.

frequently asked question (FAQ) 1. A question that's asked frequently, particularly on Usenet newsgroups or other online discussion forums. 2. A document that collects many FAQs (meaning 1) and their answers.

FSF See *Free Software Foundation (FSF)*.

FTP See *File Transfer Protocol (FTP)*.

FTP active mode A mode of operation in which the FTP client initiates a command connection to the server, and the server then initiates a data connection to the client. This is the default mode of operation for most FTP sessions.

FTP passive mode A mode of operation in which the FTP client initiates both the command and the data connections to the server. This mode of operation works better than active mode with some firewall configurations.

full duplex A mode of communication in which data can be transferred in two directions at the same time.

full recursive DNS lookup A method of name resolution in which one DNS server queries a series of other DNS servers, each of which has information on more and more specific networks, in order to locate the IP address associated with a hostname.

G

gateway A computer that functions as a router between two networks.

GB See *gigabyte (GB)*.

GDM See *GNOME Display Manager (GDM)*.

General Public License (GPL) A popular open source software license. The Linux kernel and many important Linux programs have been released under the GPL.

GID See *group ID (GID)*.

gigabit Ethernet A variety of Ethernet that can transfer 1,000 megabits (1 gigabit) per second.

gigabyte (GB) A measure of data size, most commonly meaning 2^{30} (1,073,741,824) bytes. Although this meaning is common in the computer field, *gigabyte* more technically means 10^9 (1,000,000,000) bytes, with 2^{30} bytes being more properly called “gibibyte” (GiB).

glibc A specific type of C library used on Linux systems since the late 1990s.

Globally Unique Identifier (GUID) A 16-byte number that’s intended to be a unique identifier of a specific data structure or data type—for instance, of a partition on a hard disk or of the filesystem type it contains. GUIDs are commonly used in disk filesystems, in disk partitions, and in certain network applications.

GMT See *Greenwich Mean Time (GMT)*.

GNOME Display Manager (GDM) A popular XDMCP server for Linux.

GNU Recursive acronym for GNU’s Not Unix. GNU is a Free Software Foundation (FSF) project whose goal is to build an entirely open source OS that works like Unix. The term is also used by some non-FSF projects.

GNU/Linux Generic term for a complete Linux OS to distinguish the complete OS from the kernel alone. This term is favored by Debian; most other distributions use “Linux” alone.

GNU Privacy Guard (GPG) Cryptographic software that permits encrypting and decrypting files or “signing” them so that a recipient can verify that they came from the claimed sender and have not been altered.

GPG See *GNU Privacy Guard (GPG)*.

GPL See *General Public License (GPL)*.

GPT See *GUID partition table (GPT)*.

Grand Unified Boot Loader (GRUB) A popular boot loader for Linux. GRUB can boot a Linux kernel or redirect the boot process to another boot loader in a non-Linux partition, thus booting other OSs. Similar to the competing Linux Loader (LILO). See also *boot loader*.

graphical user interface (GUI) A method of human/computer interaction characterized by a graphical display, a mouse to move a pointer around the screen, and the ability to perform actions by pointing at objects on the screen and clicking a mouse button.

Greenwich Mean Time (GMT) The time in Greenwich, England, unadjusted for daylight savings. Linux systems use this time internally and adjust to local time by knowing the system’s time zone.

group A collection of users. Files are owned by a user and a group, and group members may be given access to files independent of the owner and all other users. This feature may be used to enhance collaborative abilities by giving members of a group read/write access to particular files, while still excluding those who aren’t members of the group. It can also be used by system administrators to control access to system files and resources.

group administrator A person with administrative authority over a group. A group administrator can add or delete members from a group and perform similar administrative tasks.

group ID (GID) A number associated with a particular group. Similar to a user ID (UID).

group owner The group with which a file is most strongly associated, after the file owner.

GRUB See *Grand Unified Boot Loader (GRUB)*.

GUI See *graphical user interface (GUI)*.

GUID See *Globally Unique Identifier (GUID)*.

GUID partition table (GPT) A type of partitioning scheme that supports disks greater than 2 terabytes (2TB) in size. GPT is the likely successor to the MBR partitioning scheme that’s been dominant on x86 and x86-64 computers since the 1980s.

H

hacker 1. An individual who is skilled at using or programming computers and who enjoys using these skills in constructive ways. Many Linux programmers consider themselves hackers in this sense of the term. 2. A cracker (see also *cracker*). This use of the term is more prevalent in the mass media, but it is frowned upon in the Linux community.

half-duplex A type of data transmission in which data can be sent in only one direction at a time.

hard link A directory entry for a file that has another directory entry. All hard links are equally valid ways of accessing a file, and all must be deleted in order to delete a file. See also *soft link*.

hardware address A code that uniquely identifies a single network interface. This address is built into the device itself rather than assigned in Linux.

hash An encryption method in which a file or string is encoded in a manner that cannot be reversed. Hashes are typically much shorter than the files used to create them. Hashes are commonly used for password storage and as a more secure variant on checksums, among other things. See also *checksum*.

HDD An acronym for *hard disk drive*.

header files Files that contain interface definitions for software routines contained in a library. Program source code that uses a library must refer to the associated header files.

High-Performance Parallel Interface (HIPPI) A type of network hardware that supports speeds of up to 1600Mbps over fiber-optic cabling.

HIPPI See *High-Performance Parallel Interface (HIPPI)*.

home directory A directory associated with an account, in which the user's files reside.

hostname A computer's human-readable name, such as `persephone.example.com`.

hot standby An optional feature of RAID arrays in which a spare drive may be automatically activated by the software if it detects that one of the main drives has failed.

hot swapping Adding or removing hardware while the computer is turned on.

HOWTO documents Linux documentation that describes how to accomplish some task or use a particular program. HOWTOs are usually tutorial in nature. They're archived at <http://tldp.org>, and all major distributions ship with them as well.

HTTP See *Hypertext Transfer Protocol (HTTP)*.

HTTPS See *Hypertext Transfer Protocol Secure (HTTPS)*.

hub A type of network hardware that serves as a central exchange point in a network. Each computer has a cable that links to the hub, so all data pass through the hub. Hubs echo all data they receive to all the other computers to which they connect. See also *switch*.

hung Term used to describe a program that has stopped responding to user input, network requests, or other types of input to which it should respond. Hung processes sometimes consume a great deal of CPU time.

Hypertext Transfer Protocol (HTTP) A protocol used for transferring Web pages from a Web server to a Web browser.

Hypertext Transfer Protocol Secure (HTTPS) A variant of HTTP that enables Secure Sockets Layer (SSL) encryption.

I

IEEE-1394 An external bus technology that's used to connect high-speed external devices such as hard disks, scanners, and video equipment.

IMAP See *Internet Message Access Protocol (IMAP)*.

incremental backup A type of backup in which only files that have changed since the last backup are backed up. This is used to reduce the time required to back up a computer, at the cost of potentially greater restoration complexity.

Industry Standard Architecture (ISA) The expansion bus used on the original IBM PC. Most manufacturers began dropping ISA from their motherboards around 2001.

info pages A type of documentation similar to man pages (see *man pages*), but with a more complex hyperlinked structure within each document. The FSF and some other developers now favor info pages over man pages.

inode A filesystem (meaning 1) data structure that contains critical information on the file, such as its size and location on the disk.

input/output (I/O) A term that describes the acceptance of data from an external source or the sending of data to an external source. In some cases, the “external source” may be internal to the computer, as in I/O between a hard disk and the CPU or memory. In other cases, I/O is more clearly external, as in network I/O.

installed file database A database of files installed via the computer's package manager (such as RPM or Debian), as well as associated information such as dependencies. Also called the “package database.”

internal transfer rate The rate of data transfer within a device. This is typically applied to hard disks and similar devices to describe how quickly they can read or write data from their physical media.

International Organization for Standardization (ISO) An international nongovernmental organization that publishes technical standards in many fields. In the computer field, examples include ISO-9660 (a common CD-ROM filesystem) and ISO/IEC 26300:2006 (the OpenDocument Format for office files, implemented by OpenOffice.org and several other Linux productivity programs).

internet Any collection of networks linked together by routers. See also *Internet*.

Internet The largest network on Earth, which connects computers from around the globe. When used in this way, the word is always capitalized. See also *internet*.

Internet Message Access Protocol (IMAP) A protocol for exchanging mail messages. The recipient initiates an IMAP session. IMAP differs from POP in that IMAP enables the recipient to leave messages in organized folders on the server; POP requires that the recipient download the messages to organize them.

Internet Packet Exchange (IPX) A protocol that underlies much of Novell's original networking protocols. Despite the name, this protocol is unrelated to the Internet.

Internet Printing Protocol (IPP) A protocol for printing on a network, used in Linux by the Common Unix Printing System (CUPS) server.

Internet Protocol (IP) A low-level portion of the TCP/IP stack responsible for handling network addressing. See also *IPv4* and *IPv6*.

Internet Software Consortium (ISC) See *Internet Systems Consortium (ISC)*.

Internet Systems Consortium (ISC) A nonprofit corporation that develops a number of important network servers, including BIND, the standard Linux DHCP server, and an NTP server.

interrupt request (IRQ) A method by which peripherals (SCSI host adapters, sound cards, and so on) signal that they require attention from the CPU. An IRQ also refers to a specific interrupt signal line. The basic 32-bit x86 architecture supports 16 IRQs, numbered 0–15, but IRQs 2 and 9 are linked, so in practice, there are only 15 IRQs in the traditional architecture, and many of these are used by basic hardware like floppy disks. Modern computers usually support more than the 15 IRQs of the traditional x86 architecture.

intrusion detection system (IDS) Software that can detect suspicious activity on a computer or network and alert an operator to this activity.

I/O See *input/output (I/O)*.

IP See *Internet Protocol (IP)*.

IP address A computer's numeric TCP/IP address, such as 192.168.45.203.

IPP See *Internet Printing Protocol (IPP)*.

IPv4 The version of the Internet Protocol that's in common use in 2009. IPv4 supports 32-bit addresses, typically represented as four decimal numbers separated by dots, as in 192.168.45.203. The IPv4 address space is too small for continued expansion, so a transition to IPv6 is underway. See also *IPv6*.

IPv6 The “next-generation” Internet Protocol, which uses 128-bit addresses. This upgrade to TCP/IP supports a theoretical maximum of approximately 3.4×10^{38} addresses, as opposed to the 4 billion addresses possible with IPv4. IPv6 addresses are represented as a series of eight 2-byte hexadecimal numbers separated by colons, as in fed1:0db8:85a3:08d3:1319:8a2e:0370:7334.

IPX See *Internet Package Exchange (IPX)*.

IRQ See *interrupt request (IRQ)*.

ISA See *Industry Standard Architecture (ISA)*.

ISC See *Internet System Consortium (ISC)* (formerly *Internet Software Consortium*).

ISO See *International Organization for Standardization (ISO)*.

J

Java A programming language created by Sun Microsystems. Java is used on devices as diverse as cell phones and large server computers.

Java Virtual Machine (JVM) A method of running cross-platform computer software written in the Java programming language.

JFS See *Journalized Filesystem (JFS)*.

Journalized Filesystem (JFS) One of several journaling filesystems for Linux. JFS was developed by IBM for its AIX OS. A subsequent implementation was created for OS/2, and Linux's JFS is derived from this code.

journaling filesystem A type of filesystem that maintains a record of its operations. Such filesystems can typically recover quickly after a power failure or system crash. Common Linux journaling filesystems are ext3fs, ext4fs, ReiserFS, JFS, and XFS. See also *filesystem*.

JVM See *Java Virtual Machine (JVM)*.

K

KDE Display Manager (KDM) A popular XDMCP server for Linux.

KDM See *KDE Display Manager (KDM)*.

kernel The core program of any OS. The kernel provides interfaces between the software and the hardware and controls the operation of all other programs. Technically, the Linux kernel is the *only* component that is Linux; everything else, such as shells, X, and libraries, is available on other Unix-like systems.

kernel module A driver or other kernel-level program that may be loaded or unloaded as required.

kernel module autoloader A utility that loads and unloads kernel modules as required by the kernel, obviating the need to manually load and unload kernel modules.

kernel ring buffer A record of recent messages generated by the Linux kernel. Immediately after a Linux system boots, this buffer contains the bootup messages generated by drivers and major kernel subsystems. This buffer may be viewed with the `dmesg` command.

L

L2TP See *Layer 2 Tunneling Protocol (L2TP)*.

Layer 2 Tunneling Protocol (L2TP) A method of passing multiple network protocols over a single connection, as in a VPN.

LBA See *linear block addressing (LBA)*.

LDAP See *Lightweight Directory Access Protocol (LDAP)*.

libc See *C library (libc)*.

library A collection of code that's potentially useful to many programs. This code can be stored in files separate from the programs that use the library to save disk space and RAM when running those programs.

Lightweight Directory Access Protocol (LDAP) A tool for storing data in a type of database and accessing it over a network. LDAP is sometimes used as a form of network-based authentication.

LILO See *Linux Loader (LILO)*.

linear block addressing (LBA) A method of accessing data on a disk that uses a single sector number to retrieve data from that sector. LBA contrasts with cylinder/head/sector (CHS) addressing. Some sources refer to LBA as *logical* block addressing.

Linux 1. The open source kernel designed by Linus Torvalds as the core of a Unix-like operating system (OS). 2. A complete OS built around Linus Torvalds's kernel. See also *GNU/Linux*.

Linux Loader (LILO) A popular Linux boot loader. LILO can boot a Linux kernel or redirect the boot process to another boot loader in a non-Linux partition, thus booting other OSs. LILO is similar to the competing Grand Unified Boot Loader (GRUB). See also *boot loader*.

load average A measure of the demands for CPU time by running programs. A load average of 0 means no demand for CPU time, 1 represents a single program placing constant demand on the CPU, and values higher than 1 represent multiple programs competing for CPU time. The `top` and `uptime` commands both provide load average information.

LocalTalk A type of network hardware common on older Macintosh networks.

log file A text file maintained by the system as a whole or an individual program, in which important system events are recorded. Log files typically include information on user logins, server access attempts, and automatic routine maintenance.

log file rotation See *log rotation*.

log rotation A routine maintenance process in which the computer suspends recording data in log files, renames them, and opens new log files. This process keeps log files available for a time, but ultimately it deletes them, preventing them from growing to consume all available disk space.

logical block addressing (LBA) See *linear block addressing (LBA)*.

logical partition A type of MBR hard disk partition that has no entry in the primary partition table. Instead, logical partitions are carried within an extended partition.

logical volume management (LVM) An alternative or supplement to the normal partitioning system, in which partition substitutes (logical volumes) may be created, deleted, or resized dynamically, much like files on a filesystem. LVM simplifies volume management by eliminating the need to plan where volumes reside on a disk, but at the cost of an extra layer of data structures.

loop A programming or scripting construct enabling multiple executions of a segment of code. Typically terminated through the use of a conditional expression.

LVM See *logical volume management (LVM)*.

M

MAC See *mandatory access control (MAC)* or *media access control (MAC) address*.

MAC address See *media access control (MAC) address*.

machine name The portion of a hostname that identifies a computer on a network, as opposed to the network as a whole (for instance, `ginkgo` is the machine name portion of `ginkgo.example.com`). The machine name is sometimes used in reference to the entire hostname.

mail exchanger (MX) In DNS configuration, refers to a computer that should accept mail for a domain. For instance, if the MX for `example.com` is `franklin.example.com`, then mail addressed to `ben@example.com` will be sent to `franklin.example.com`. This computer

will then deliver the mail to the local user **ben**, forward it to another account or computer, or bounce the mail.

mail transfer agent (MTA) A mail server program or computer.

mail user agent (MUA) A mail reader (client) program.

main memory The main type of RAM in a computer, as opposed to cache memory.

major version number The first number in a program's version number. For instance, if a program's version number is 1.2.3, the major version number is 1.

man pages An electronic "manual" for a program, configuration file, system call, or other feature of the system. Man pages are accessed by typing **man** followed by the program or other topic you want to learn about, as in **man man** to learn about the man pages system itself.

mandatory access control (MAC) A security tool employed by SELinux to constrain who may perform particular actions.

master One of two PATA devices on a single PATA chain. The master device gets a lower Linux device letter than the slave device does.

master boot record (MBR) The first sector of a hard disk. The MBR contains code that the BIOS runs during the boot process, as well as the primary partition table. *MBR* sometimes refers to the MBR partitioning scheme itself. See also *GUID partition table (GPT)*.

MB See *megabyte (MB)*.

MBR See *master boot record (MBR)*.

MD4 password See *Message Digest 4 (MD4) password*.

MD5 password See *Message Digest 5 (MD5) password*.

media access control (MAC) address A low-level address associated with a piece of network hardware. The MAC address is usually stored on the hardware itself, and it is used for local network addressing only. Addressing between networks (such as on the Internet) uses higher-level addresses, such as an IP address.

megabyte (MB) A measure of data size, most commonly meaning 2^{20} (1,048,576) bytes. Although this meaning is common in the computer field, *megabyte* more technically means 10^6 (1,000,000) bytes, with 2^{20} bytes being more properly called "mebibyte" (MiB).

Message Digest 4 (MD4) password A password stored using the Message Digest 4 (MD4) hash. MD4 passwords are common on Windows systems and are also used by Samba's encrypted password system.

Message Digest 5 (MD5) password A password that's stored using the Message Digest 5 (MD5) hash. Recent Linux systems generally use MD5 or SHA1 passwords.

mode The permissions of a file. In conjunction with the file's owner and group, the mode determines who may access a file and in what ways.

mode lines Definition of the timings required by particular video resolutions running at particular refresh rates.

modem This word is short for “modulator/demodulator.” It’s a device for transferring digital data over an analog transmission medium. Traditionally, the analog transmission medium has been the normal telephone network, but the word “modem” is frequently applied to devices used for broadband Internet access as well.

module A driver or other software component that’s stored in a separate file. The Linux kernel supports modules, as do some other software packages, such as Apache. Software designed to use modules can load them on demand or on command, saving RAM when modules aren’t in use and reducing the size of the main program.

motherboard The main circuit board in a computer. The CPU, RAM, and add-on cards typically plug directly into the motherboard, although some designs place some of these components on extender cards. The motherboard is also sometimes referred to as the “mainboard” or the “system board.”

mount 1. The process of adding a filesystem (meaning 1) to a directory tree. 2. A command of the same name that performs this task.

mount point A directory to which a new filesystem (meaning 1) is attached. Mount points are typically empty directories before their host filesystems are mounted.

MTA See *mail transfer agent (MTA)*.

MUA See *mail user agent (MUA)*.

MX See *mail exchanger (MX)*.

N

Name Service Cache Daemon (NSCD) A daemon that caches common name service requests, such as those for usernames and hostnames.

NDP See *Neighbor Discovery Protocol (NDP)*.

Neighbor Discovery Protocol (NDP) A protocol used to learn a network hardware address based on an IPv6 address. See also *Address Resolution Protocol (ARP)*.

NetBEUI A network stack similar to AppleTalk or TCP/IP in broad outline but used primarily on local networks.

NetBIOS Networking protocols that are often used in conjunction with NetBEUI or TCP/IP. NetBIOS underlies the SMB/CIFS file sharing protocols used by Microsoft Windows and implemented in Linux by Samba.

netmask See *network mask*.

Network Filesystem (NFS) A file sharing protocol used among Linux and Unix computers.

Network Information Service (NIS) A network protocol that enables computers to share simple database files. Commonly used to provide centralized login authentication and as a substitute for DNS on small networks.

network interface card (NIC) The hardware that provides the network circuitry for a computer. Traditionally, NICs have been plug-in ISA or PCI cards; however, most modern computers implement network circuitry on their motherboards.

network mapper (Nmap) A common text-based network port scanner for Linux.

network mask A bit pattern that identifies the portion of an IP address that's an entire network and the part that identifies a computer on that network. For IPv4 addresses, the pattern may be expressed as 4 decimal bytes separated by dots (as in 255.255.255.0) or as the number of network bits following an IP address and a slash (as in 192.168.45.203/24). The network mask is also referred to as the "netmask" or "subnet mask."

Network News Transfer Protocol (NNTP) A network protocol that widely disseminates posts in any of hundreds of network newsgroups. See also *Usenet*.

New Technology File System (NTFS) The default filesystem for modern versions of Windows (NT/200x/XP/Vista). Several NTFS drivers for Linux exist, but none is perfect.

NFS See *Network Filesystem (NFS)*.

NIC See *Network Interface Card (NIC)*.

NIS See *Network Information Service (NIS)*.

Nmap See *Network Mapper (Nmap)*.

NNTP See *Network News Transfer Protocol (NNTP)*.

non-volatile RAM (NVRAM) A type of memory that retains data even after power is cut off. NVRAM is commonly used to store BIOS settings.

NSCD See *Name Service Cache Daemon (NSCD)*.

O

open mail relay An SMTP mail server that's configured to relay mail from anywhere to anywhere. Open mail relays can be abused by spammers to obfuscate their messages' true origins.

open port A network port that's being used by a server program and that's accessible by outside systems. Ports that are open unnecessarily pose a security risk and should be closed.

open source Copyrighted software (or other works, such as documentation) that may be freely redistributed under terms laid out at <http://opensource.org/docs/osd>. The Linux

kernel and all software required to make a complete Linux distribution are open source, although Linux can also run proprietary software.

Open System Interconnection (OSI) model A means of describing network stacks, such as TCP/IP, NetBEUI, or AppleTalk. In the OSI model, such stacks are broken down into several layers, each of which communicates directly with the layers above and below it.

OS Acronym for *operating system*.

OSI model See *Open System Interconnection (OSI) model*.

P

package database See *installed file database*.

packet A limited amount of data collected together with an envelope and sent over a network. See also *envelope*.

packet filter firewall A type of firewall that operates on individual network data packets, passing or rejecting packets based on information such as the source and destination addresses and ports.

packet sniffer A program that monitors network traffic at a low level, enabling diagnosis of problems and capturing data. Packet sniffers can be used both for legitimate network diagnosis and for data theft.

PAM See *Pluggable Authentication Modules (PAM)*.

Parallel ATA (PATA) The traditional form of ATA interface, in which several bits are transferred at once. See also *Serial ATA (SATA)*.

parameter An option passed to a program on a command line or as part of a configuration file.

parent process A relative term referring to the process that started another. For instance, if you launch an editor from a shell, the shell process is the editor's parent process.

parent process ID (PPID) A PID number associated with the parent of a process.

partition A contiguous part of a hard disk that's set aside to hold a single filesystem (meaning 1). Also used as a verb to describe the process of creating partitions on a hard disk.

partition table The disk data structure that describes the layout of partitions on a hard disk. See also *master boot record (MBR)* and *GUID partition table (GPT)*.

PATA See *Parallel ATA (PATA)*.

path A colon-delimited list of directories in which program files may be found. (Similar lists define the locations of directories, fonts, and other file types.)

payload The portion of a network data packet that contains the actual data to be transmitted, as opposed to the envelope.

PCI See *Peripheral Component Interconnect (PCI)*.

PCL See *Printer Control Language (PCL)*.

peripheral A device that connects to and is controlled by a computer. Many peripherals, such as Web cams and keyboards, are external to the computer's main box. Some definitions include devices that reside within the computer's main box, such as hard disks and CD-ROM drives.

Peripheral Component Interconnect (PCI) An expansion bus capable of much higher speeds than the older ISA bus. Modern computers usually include several PCI slots.

permission bit A single bit used to define whether a given user or class of users has a particular type of access to a file. For instance, the owner's execute permission bit determines whether the owner can run a file as a program. The permission bits together comprise the file's mode.

Personal Home Page (PHP) See *PHP: Hypertext Preprocessor (PHP)*.

phishing The process of sending bogus e-mail or putting up fake Web sites with the goal of collecting sensitive personal information (typically credit card numbers).

PHP See *PHP: Hypertext Preprocessor (PHP)*.

PHP: Hypertext Preprocessor (PHP) A recursive acronym referring to a scripting language for Web servers that enables them to customize Web pages for particular users or purposes. *PHP* is sometimes expanded as *Personal Home Page*.

PID See *process ID (PID)*.

PIO See *Programmed Input/Output (PIO)*.

pipe A method of executing two programs so that one program's output serves as the second program's input. Piped programs are separated in a Linux shell by a vertical bar (`|`).

pipeline See *pipe*.

Pluggable Authentication Modules (PAM) Linux's user authentication subsystem. PAM is highly configurable; you can create stacks of modules that are called for any given authentication service (such as a network server or login program), enabling you to customize the authentication database used and other tasks that are performed during a login attempt.

Point-to-Point Protocol (PPP) A method of initiating a TCP/IP connection between two computers over an RS-232 serial line or modem.

port number A number that identifies the program from which a data packet comes or to which it's addressed. When a program initiates a network connection, it associates itself with one or more ports, enabling other computers to uniquely address the program.

Post Office Protocol (POP) A mail server protocol in which the recipient initiates transfer of messages. POP differs from IMAP in that POP doesn't provide any means for the recipient to organize and store messages on the server.

PostScript A programming language used on many high-end printers. PostScript is optimized for displaying text and graphics on the printed page. The Linux program Ghostscript converts from PostScript to bitmapped formats understood by many low-end and mid-range printers.

PostScript Printer Definition (PPD) A configuration file that provides information on a printer's capabilities—its paper size, whether it prints in color, and so on.

PowerPC See *Power Performance Computing (PowerPC or PPC)*.

Power Performance Computing (PowerPC or PPC) A CPU architecture created by Motorola, IBM, and Apple in the early 1990s. PowerPC CPUs were at the heart of Macintosh computers from 1994 to 2006 (Apple has since moved to x86-64 CPUs). Today they're used mostly in certain specialized devices.

PPC See *Power Performance Computing (PowerPC or PPC)*.

PPD See *PostScript Printer Definition (PPD)*.

PPID See *parent process ID (PPID)*.

PPP See *Point-to-Point Protocol (PPP)*.

primary boot loader The first boot loader run by the BIOS.

primary partition A type of MBR partition that's defined in a data structure contained in the hard disk's partition table in the MBR. The MBR partitioning scheme supports only four primary partitions per hard disk.

print queue A storage place for files waiting to be printed.

Printer Control Language (PCL) A language developed by Hewlett-Packard for controlling printers. (Many of Hewlett-Packard's competitors also use PCL.) PCL is most commonly found on midrange laser printers, but some inkjet printers also support the language. Several PCL variants exist, the most common ranging from PCL 3 to PCL 6.

printer driver A software component that converts printable data generated by an application into a format that's suitable for a specific model of printer. In Linux, printer drivers usually reside in Ghostscript, but some applications include a selection of printer drivers to print directly to various printers.

privileged port A port (see *port number*) that's numbered below 1024. Linux restricts access to such ports to `root`. In computing's early days, any server running on a privileged port could be considered trustworthy, because only programs configured by professional system administrators could be run on such ports. Today, that's no longer the case. See also *unprivileged port*.

process A piece of code that's maintained and run by the Linux kernel separately from other pieces of code. Most processes correspond to programs that are running. One program can be run multiple times, resulting in several processes.

process ID (PID) A number associated with a specific process. Utilities such as `kill` and `renice` work on PIDs, enabling you to terminate them or alter their priorities.

Programmed Input/Output (PIO) A method of data transfer between memory and expansion cards in which the CPU actively performs the transfer. PIO tends to consume much more CPU time than DMA does.

protocol stack A collection of drivers, kernel procedures, and other software that implements a standard means of communicating across a network. Two computers must support compatible protocol stacks to communicate. The most popular protocol stack today is TCP/IP.

pull mail protocol A mail protocol in which the recipient initiates the transfer. Examples include POP and IMAP.

push mail protocol A mail protocol in which the sender initiates the transfer. SMTP is the most common push mail protocol.

R

RADIUS See *Remote Authentication Dial-In User Services (RADIUS)*.

RAID See *redundant array of independent disks (RAID)*.

random access A method of access to a storage device (RAM, hard disk, and so on) in which information may be stored or retrieved in an arbitrary order with little or no speed penalty. See also *sequential access*.

RDP See *Remote Desktop Protocol (RDP)*.

redirection A procedure in which a program's standard output or standard error is sent to a file rather than to the screen, or in which the program's standard input is obtained from a file rather than from the keyboard. See also *standard input*, *standard error*, and *standard output*.

redundant array of independent disks (RAID) Two or more disks that are treated as a single physical hard disk. RAID can improve speed, reliability, or both, depending on how it's configured. It can be implemented in special hardware RAID controllers or via special kernel options and Linux configuration.

regular expression A method of matching textual information that may vary in important ways but that contains commonalities. The regular expression captures the commonalities and uses various types of wildcards to match variable information.

ReiserFS One of several journaling filesystems for Linux. ReiserFS was developed from scratch for Linux.

relative directory name A directory name that's specified relative to the current directory. Relative directory names often include the parent specification (`..`), which indicates the current directory's parent.

release number See *build number*.

Remote Authentication Dial-In User Services (RADIUS) A network authentication tool that's often used by network gateway systems that require authentication, such as PPP servers or Wi-Fi routers.

Remote Desktop Protocol (RDP) A network protocol used by Microsoft to enable remote use of a Windows computer's GUI environment. The `rdesktop` program is a Linux RDP client.

Remote Frame Buffer (RFB) A cross-platform remote GUI login protocol used by VNC.

remote login server A type of server that enables individuals at distant locations to use a computer. Examples include Telnet, SSH, and XDM.

Request for Comments (RFC) An Internet standards document. RFCs define how protocols like Telnet and SMTP operate, thus enabling tools developed by different companies or individuals to interoperate.

RFB See *Remote Frame Buffer (RFB)*.

RFC See *Request for Comments (RFC)*.

root directory The directory that forms the base of a Linux filesystem (meaning 2). All other directories are accessible from the root directory, either directly or via intermediate directories.

root DNS servers A set of DNS servers that deliver information to other DNS servers about top-level domains (`.com`, `.net`, `.us`, and so on). DNS servers consult the root DNS servers first when performing full recursive DNS lookups.

root filesystem The filesystem (meaning 1) on a Linux system that corresponds to the root directory, along with several directories based on it.

root kit A set of scripts and other software that enable script kiddies to break into computers.

root name servers See *root DNS servers*.

root partition The partition associated with the root filesystem.

rooted An adjective describing a computer that has been compromised to the point where the intruder has full `root` access to the system.

router A computer that transfers data between networks. See also *gateway*.

RPM See *RPM Package Manager (RPM)*.

RPM Package Manager (RPM) A package file format and associated utilities designed by Red Hat but now used on many other distributions as well. RPM features excellent dependency tracking and easy installation and removal procedures.

runlevel A number associated with a particular set of services that are being run. Changing runlevels changes services or can shut down or restart the computer.

S

Samba Web Administration Tool (SWAT) A server that enables administrators to configure Samba servers from the same or another computer by using an ordinary Web browser.

SAN See *storage area network (SAN)*.

SAS See *Serial Attached SCSI (SAS)*.

SATA See *Serial ATA (SATA)*.

scp The secure copy (scp) program uses the SSH protocol to copy files over a network using encryption. It works much like the local cp program, but with extensions to enable specification of a remote server, username, and password.

script kiddy An individual with little knowledge or skill who breaks into computers using scripts created by others. Such break-ins often leave obvious traces, and script kiddies frequently cause collateral damage that produces system instability.

scripting language An interpreted computer programming language designed for writing small utilities to automate simple but repetitive tasks. Examples include Perl, Python, Tcl, and shell scripting languages like those provided by Bash and tcsh.

SCSI See *Small Computer System Interface (SCSI)*.

Second Extended Filesystem (ext2 or ext2fs) The most common filesystem (meaning 1) in Linux from the mid-1990s through approximately 2001.

secondary boot loader A boot loader that's launched by another boot loader.

Secure File Transfer Protocol (SFTP) A combination of SSH and FTP; this protocol employs an SSH link to provide encrypted transfer of files in an FTP-like way.

Secure Hash Algorithm (SHA) A type of encryption algorithm that may be used in generating checksum-like verifications of file integrity or in encrypting passwords, among other uses. Specific subtypes include SHA0, SHA1, and SHA2, with SHA1 being the most widely used in Linux.

Secure Shell (SSH) A remote login protocol and program that uses encryption to ensure that intercepted data packets cannot be used by an interloper.

Secure Sockets Layer (SSL) An encryption protocol that permits secure two-way communication over a network. Commonly used on the Web, in the form of HTTPS transfers. See also *Transport Layer Security (TLS)*.

Security Enhanced Linux (SELinux) A set of extensions to the Linux kernel and support programs that enhance security.

SELinux See *Security Enhanced Linux (SELinux)*.

Sequenced Packet Exchange (SPX) Part of the Novell networking stack, along with IPX.

sequential access A method of accessing a storage medium that requires reading or writing data in a specific order. The most common example is a tape; to read data at the end of a tape, you must wind past the interceding data. See also *random access*.

Serial ATA (SATA) A type of ATA interface that uses serial data transfer rather than the parallel data transfers used in older forms of ATA. See also *parallel ATA (PATA)*.

Serial Attached SCSI (SAS) A type of SCSI interface that uses serial data transfer rather than the parallel data transfers used in older forms of SCSI.

server 1. A program that responds to data transfer requests using networking protocols. 2. A computer that runs one or more server programs.

Server Message Block (SMB) A file sharing protocol common on Windows-dominated networks. SMB is implemented in Linux via the Samba suite. Also known as the Common Internet Filesystem (CIFS).

Server Message Block File System (SMBFS) A Linux client implementation of SMB, enabling Linux to mount remote Windows or Samba shares as if they were local disk partitions.

server program See *server*, meaning 1.

service set ID (SSID) A name that identifies a specific Wi-Fi network to distinguish it from other nearby networks.

set group ID (SGID) A special type of file permission used on program files to make the program run with the permissions of its group. (Normally, the user's group permissions are used.)

set user ID (SUID) A special type of file permission used on program files to make the program run with the permissions of its owner, rather than those of the user who runs the program.

SFTP See *Secure File Transfer Protocol (SFTP)*.

SGID See *set group ID (SGID)*.

sh A program or link in `/bin` that provides a default shell. On Linux systems, `/bin/sh` typically links to `/bin/bash`, the Bash binary.

SHA See *Secure Hash Algorithm (SHA)*.

shadow password A method of storing encrypted passwords separately from most other account information. This allows the passwords to reside in a file with tighter security options than the rest of the account information, which improves security when compared to storing all the account information in one file with looser permissions.

share In file sharing protocols, and particularly in SMB/CIFS, a named network resource associated with a directory or printer that's being shared. May also be used as a verb to describe the process of making the share available.

shell A program that provides users with the ability to run programs, manipulate files, and so on.

shell script A program written in a language that's built into a shell.

signal In reference to processes, a signal is a code that the kernel uses to control the termination of the process or to tell it to perform some task. Signals can be used to kill processes.

Simple Mail Transfer Protocol (SMTP) The most common push mail protocol on the Internet. SMTP is implemented in Linux by servers such as sendmail, Postfix, Exim, and qmail.

Simple Network Management Protocol (SNMP) A protocol for reporting on the status of a computer over a network or for adjusting a computer's settings remotely.

slave The second of two possible devices on a PATA chain. The slave device has a higher Linux device letter than the master device does.

Small Computer System Interface (SCSI) An interface standard for hard disks, CD-ROM drives, tape drives, scanners, and other devices.

smart filter A program, run as part of a print queue, that determines the type of a file and passes it through appropriate programs to convert it to a format that the printer can handle.

SMB See *Server Message Block (SMB)*.

SMBFS See *Server Message Block File System (SMBFS)*.

SMTP See *Simple Mail Transfer Protocol (SMTP)*.

SNMP See *Simple Network Management Protocol (SNMP)*.

social engineering The practice of convincing individuals to disclose sensitive information without arousing suspicion. Social engineers may pretend to be system administrators to ask for passwords, for instance. See also *phishing*.

soft link A type of file that refers to another file on the computer. When a program tries to access a soft link, Linux passes the contents of the linked-to file to the program. If the linked-to program is deleted, the soft link stops working. Deleting the soft link doesn't affect the original file. Also referred to as a "symbolic link." See also *hard link*.

software modem Modems that implement key functionality in software that must be run by the host computer. These modems require special drivers, which are uncommon in Linux.

source package A file that contains complete source code for a program. The package may be compiled into a binary package, which can then be installed on the computer.

source RPM A type of source package that uses the RPM file format.

spam Unsolicited bulk e-mail.

spawn The action of one process starting another.

spool directory A directory in which print jobs, mail, or other files wait to be processed. Spool directories are maintained by specific programs, such as the printing system or SMTP mail server.

SPX See *Sequenced Packet Exchange (SPX)*.

SSH See *Secure Shell (SSH)*.

SSID See *service set ID (SSID)*.

SSL See *Secure Sockets Layer (SSL)*.

standard error The default method of delivering purely text-based error messages from a program to the user. It normally corresponds to a text-mode screen, `xterm` window, or the like. Sometimes referred to as "stderr." See also *standard output*.

standard input The default method of delivering input to a program. It normally corresponds to the keyboard at which you type. Sometimes referred to as "stdin."

standard output The default method of delivering purely text-based nonerror information from a program to the user. It normally corresponds to a text-mode screen, `xterm` window, or the like. Sometimes referred to as "stdout." See also *standard error* and *standard input*.

startup script A script that controls part of the Linux boot process.

stateful packet inspection A firewall tool in which a packet's state (that is, whether it's marked to begin a transaction, to continue an existing exchange, and so on) is considered in the filtering process.

stderr See *standard error*.

stdin See *standard input*.

stdout See *standard output*.

sticky bit A special file permission bit that's most commonly used on directories. When set, only a file's owner may delete the file, even if the directory in which it resides can be modified by others.

storage area network (SAN) A technology that permits network-attached disks and other storage devices to look just like local storage to client computers.

subdomain A subdivision of a domain. A subdomain may contain computers or subdomains of its own.

subnet mask See *network mask*.

SUID See *set user ID (SUID)*.

super server A server that listens for network connections intended for other servers and launches those servers. Examples on Linux are *inetd* and *xinetd*.

superuser A user with extraordinary rights to manipulate critical files on the computer. The superuser's username is normally *root*.

swap file A disk file configured to be used as swap space.

swap partition A disk partition configured to be used as swap space.

swap space Disk space used as an extension to a computer's RAM. Swap space enables a system to run more programs or to process larger data sets than would otherwise be possible.

SWAT See *Samba Web Administration Tool (SWAT)*.

switch A type of network hardware that serves as a central exchange point in a network. Each computer has a cable that links to the switch, so all data pass through the switch. A switch usually sends data only to the computer to which it's addressed. See also *hub*.

symbolic link See *soft link*.

system cron job A cron job that handles system-wide maintenance tasks, such as log rotation or deletion of unused files from */tmp*. See also *user cron job*.

System V (SysV) A form of AT&T Unix that defined many of the standards used on modern Unix systems and Unix clones, such as Linux.

SysV See *System V (SysV)*.

SysV startup script A type of startup script that follows the System V startup standards. Such a script starts one service or related set of services.

T

tarball A package file format based on the tar utility. Tarballs are easy to create and are readable on any version of Linux, or most non-Linux systems. They contain no dependency information, and the files they contain are not easy to remove once installed.

TCP/IP See *Transmission Control Protocol/Internet Protocol (TCP/IP)*.

Telnet A protocol used for performing remote text-based logins to a computer. Telnet is a poor choice for connections over the Internet because it passes all data, including passwords, in an unencrypted form, which is a security risk. See also *Secure Shell (SSH)*.

terminal program 1. A GUI program that's used to run text-based programs in a GUI environment. Examples include programs called xterm, Konsole, and Terminal. 2. A program that's used to initiate a simple text-mode connection between two computers, especially via a modem or RS-232 serial connection.

text editor A program for editing text files on a computer.

TFTP See *Trivial File Transfer Protocol (TFTP)*.

Third Extended Filesystem (ext3 or ext3fs) A variant of the *Second Extended Filesystem (ext2 or ext2fs)* that adds a journal to reduce startup times after a power failure or system crash. See also *journaling filesystem*.

time to live (TTL) A limited "lifetime" for computer data, used to prevent time-sensitive data from being retained or retransmitted indefinitely and thereby causing problems. TTLs are used in DNS records to ensure that DNS servers periodically check authoritative sources for changed data and in low-level data packets to ensure that the packets don't end up circling a network forever in case of a delivery error.

TLS See *Transport Layer Security (TLS)*.

Token Ring A type of network hardware that supports speeds of up to 1Gbps on twisted-pair cabling.

Transmission Control Protocol/Internet Protocol (TCP/IP) A very popular network stack and the one on which the Internet is built.

Transport Layer Security (TLS) An encryption protocol that permits secure two-way communication over a network. Serves as the successor to Secure Sockets Layer (SSL) encryption in some applications.

Triple Data Encryption Standard (3DES) A data encryption standard.

Trivial File Transfer Protocol (TFTP) A simple file transfer protocol that's most commonly used to provide files to computers, such as thin clients, that boot off of the network rather than from a local disk.

TTL See *time to live (TTL)*.

U

UID See *user ID (UID)*.

umask See *user mask (umask)*.

Unicode Transformation Format (UTF) A method of encoding letters, numbers, and punctuation as numbers. UTF is superior to the older ASCII in that it can support more alphabets, so it's better at handling non-English languages.

Universal Serial Bus (USB) A type of interface for low- to medium-speed external devices, such as keyboards, mice, cameras, modems, scanners, and removable disk drives. USB 2.0 increases the speed to the point that USB is usable for hard disks in less demanding applications.

unprivileged port A port (see *port number*) that's numbered above 1024. Such ports may be accessed by any user and so are commonly used by client programs and by a few servers that may legitimately be run by ordinary users. See also *privileged port*.

USB See *Universal Serial Bus (USB)*.

user An individual who has an account on a computer. This term is sometimes used as a synonym for *account*.

user cron job A cron job created by an individual user to handle tasks for that user, such as running a CPU-intensive job late at night when other users won't be disturbed by the job's CPU demands. See also *system cron job*.

user ID (UID) A number associated with a particular account. Linux uses the UID internally for most operations, and it converts to the associated username only when interacting with people.

user mask (umask) A bit pattern representing the permission bits that are to be removed from files created from a process.

user private group A group strategy in which every user is associated with a unique group. Users may then add other users to their groups in order to control access to files on an individual basis.

username The name associated with an account, such as *theo* or *miranda*. Linux usernames are case sensitive and may be from 1 to 32 characters in length, although they're usually entirely lowercase and no longer than 8 characters.

UTC See *Coordinated Universal Time (UTC)* and *Greenwich Mean Time (GMT)*.

Usenet A distributed set of computers that host online discussions on a wide array of topics in hundreds of different news groups, such as *comp.os.linux.misc* for Linux discussions. Usenet predates the Web forums that have largely eclipsed it in the public eye, but Usenet remains a valuable resource for learning about Linux—and many other things.

V

variable In computer programming or scripting, a “placeholder” for data. Variables may change from one run of a program to another, or even during a single run of a program.

VFAT See *Virtual FAT (VFAT)*.

Virtual FAT (VFAT) A variant of the FAT filesystem that supports long filenames. Microsoft introduced VFAT with Windows 95, and most modern OSs now support it. To use a VFAT filesystem in Linux, mount it with the `vfat` filesystem type code.

virtual filesystem A filesystem that doesn’t correspond to a real disk partition, removable disk, or network export. A common example is `/proc`, which provides access to information on the computer’s hardware.

virtual hosting A process by which a single computer can host servers (particularly Web servers) for multiple domains. For instance, one computer might respond to the names `www.pangaea.edu`, `www.example.com`, and `web.littrow.luna.edu`, delivering different content for each name.

Virtual Network Computing (VNC) A cross-platform remote GUI login tool based on the RFB protocol. VNC clients and servers are available for Linux, Windows, Mac OS, and many other platforms.

Virtual Private Network (VPN) A tool for connecting two network segments across a potentially insecure network, such as the Internet. VPNs can enable secure communications between offices or between individuals and remote networks.

virtual terminal (VT) One of several independent text-mode or GUI screens maintained by Linux. You can log in multiple times and run different programs in each VT and then switch between them by pressing `Ctrl+Alt+Fn`, where *n* is the terminal number (such as `Ctrl+Alt+F4` to switch to VT 4).

VNC See *Virtual Network Computing (VNC)*.

VPN See *Virtual Private Network (VPN)*.

VT See *virtual terminal (VT)*.

W

WAP See *wireless access point (WAP)*.

WEP See *Wired Equivalent Privacy (WEP)*.

Wi-Fi A common name for wireless networking using any of the *802.11* standards.

Wi-Fi Protected Access (WPA or WPA2) An encryption protocol used on wireless networks. Both WPA and WPA2 are superior to the older WEP, and WPA2 is superior to WPA.

wildcard A character or group of characters that, when used in a shell as part of a filename, match more than one character. For instance, `b??k` matches `book`, `back`, and `buck`, among many other possibilities.

window manager A program that provides decorative and functional additions to the plain windows provided by X. Linux supports dozens of window managers.

Wired Equivalent Privacy (WEP) An encryption protocol used on wireless networks. WEP is inferior to the more recent WPA and WPA2 protocols.

wireless access point (WAP) A hardware device that serves to connect multiple computers using a wireless (Wi-Fi) network protocol. WAPs also usually connect these computers to a wired network.

wireless router A device that combines WAP, network switch, and broadband router functionality in one box.

workstation A type of computer that's used primarily by one individual at a time to perform productivity tasks, such as drafting, scientific or engineering simulations, or writing. See also *desktop computer*.

WPA or WPA2 See *Wi-Fi Protected Access (WPA or WPA2)*.

X

X Shortened form of “X Window System.”

x86 The most common CPU type for desktop computers, workstations, and small servers from the mid-1980s until about 2005. The x86 architecture is 32 bits in size (pre-80386 CPUs were only 16-bit, though). The architecture was created by Intel, but AMD and others make or made x86 CPUs. See also *x86-64*.

x86-64 A 64-bit extension to the x86 CPU architecture. This architecture was created by Advanced Micro Devices (AMD) and is used in most modern CPUs from both AMD and Intel. These CPUs may run both 64-bit and 32-bit code, making the upgrade from x86 to x86-64 relatively painless. Also referred to as “AMD64” and “EM64T.”

X.org-X11 A popular X server on Linux systems, starting in 2004. X.org-X11 6.7.0 forked from XFree86 4.3.99.

X client A program that uses X to interact with the user.

X Display Manager (XDM) The simplest of several common XDMCP programs for Linux.

X Display Manager Control Protocol (XDMCP) A protocol that accepts either remote or local X-based logins to a computer. Examples include XDM, KDM, and GDM.

XDM See *X Display Manager (XDM)*.

XDMCP See *X Display Manager Control Protocol (XDMCP)*.

XFS See *Extent Filesystem (XFS)*.

X server A program that implements X for a computer; especially the component that interacts most directly with the video hardware.

X Window System The GUI environment for Linux. The X Window System is a network-aware, cross-platform GUI that relies on several additional components (such as a window manager and widget sets) to provide a complete GUI experience.

XFree86 A set of X servers and related utilities for Linux and other OSs. Abandoned on most distributions in favor of X.org-X11.

xterm A very common terminal program (meaning 1).



Index

Note to the Reader: Throughout this index **boldfaced** page numbers indicate primary discussions of a topic. *Italicized* page numbers indicate illustrations.

A

- A (address) records, 445
- AAAA records, 445
- absolute directory names, 57
- Accelerated-X server, 20
- access control lists (ACLs), **83–84**, 250
- access_times option, 158
- accounts. *See* users and user accounts
- ACLs (access control lists), **83–84**, 250
- active mode in FTP, 488
- address (A) records, 445
- Address Resolution Protocol (ARP)
 - caches, 376
 - IP address conversion, 360
- addresses
 - disk sectors, 10
 - IP
 - with DHCP, **438–441**
 - overview, **359–362**
 - static, **369–373**
 - TCP/IP, 356, 358
- adduser utility, 197
- administrative logins, **191**
- administrators, group, 214
- ADS setting, 479
- Advanced Linux Sound Architecture (ALSA) project, 31
- Advanced Package Tool (APT) utilities, 304, **313**, 314, 517
- Advanced Technology Attachment (ATA), 239–241
- Aho, Alfred J., 175
- aliases for e-mail, **460–461**
- Allowed IP or network addresses
 - setting, 158
- ALSA (Advanced Linux Sound Architecture) project, 31
- altered data files as intrusion symptom, 538
- AMANDA package, 326
- ampersands (&)
 - processes, 119
 - redirection, 69
 - shell commands, 50–51, 91
 - startup files, 158–159
- Anacron package, 166
- anonymous FTP access, 483, **485**, 488
- apache package, 489
- Apache servers, **489–490**
 - controlling, **493–494**
 - modules, 490
 - options, **489–490**
- apachectl utility, **493–494**
- APM (Apple Partition Map), 242
- append RAID approach, 276–277
- appending text to files, 68
- Apple Partition Map (APM), 242
- AppleTalk protocol, 357
- apropos command, 92
- APT (Advanced Package Tool) utilities, 304, **313**, 314, 517
- apt-cache tool, 313
- apt-get commands, **311–313**
- apt-key tool, 313
- apt-sortpkgs tool, 313
- architecture codes in RPM, **299–300**
- ARKEIA package, 326

ARP (Address Resolution Protocol)
 caches, 376
 IP address conversion, 360
 arp tool, 376
 ASCII transfer FTP mode, 488
 asterisks (*)
 account passwords, 205, 536
 cron jobs, 126
 info pages, 93
 log files, 164–165
 ls command, 56
 NTP servers, 453
 port numbers, 532
 PPP servers, 375
 traceroute command, 378
 XDM servers, 412
 at command, 128–129
 at-signs (@)
 domains, 445–446
 FTP, 488
 hostnames, 169, 407
 remote machine names, 165
 SOA records, 446
 ATA (Advanced Technology
 Attachment) devices, 239–241
 atd tool, 129
 atomic clocks, 449, 450
 atq tool, 129
 atrm tool, 129
 auditing, 531
 account reviews, 535–536
 open ports, 531–535,
 534–535
 verifying installed files and
 packages, 537
 authentication
 networks, 454–455, 523–526
 overview, 520–521
 PAM configuration, 521–523
 Samba, 479

Author section in man pages, 93
 autoconf utility, 319
 autodetection of video card chipsets, 18
 Automatically Obtain IP Address
 Settings With option, 374
 automounter support, 268
 AutoRepeat option, 22
 awk tool, 175–177

B

background processes, 119
 backslash characters (\)
 directories, 55
 regular expressions, 177
 backticks (`)
 sendmail configuration, 458
 text within, 70–71
 backups
 cpio, 326–329
 exam essentials, 340
 GPT, 244
 hardware, 324–326
 network, 333–334
 partitions for, 246
 for recovery, 335–336
 restoring, 329–333
 review questions, 341–347
 scheduling, 334–335
 summary, 339
 Bash shell, 49, 52–53, 89
 batch tool, 129
 beginnings of log files, checking, 173
 Berkeley Internet Name Domain
 (BIND), 441, 443–448
 Berkeley Standard Distribution (BSD)
 disk labels, 242
 printing systems, 416, 424, 427
 ps options, 110–111
 bg program, 51, 119

- /bin directory, 247
- binary files with FTP, 488
- binary form, 89
- binary packages, 297
- BIND (Berkeley Internet Name Domain), 441, 443–448
- bind options
 - DNS, 364, 442
 - xinetd, 157
- bind-tools package, 363
- bind-utils package, 363
- BIOS
 - boot options, 4
 - CHS translation, 10
 - EFI for, 243
 - NVRAM for, 239
 - passwords, 515
 - setup screens, 7
- Blackbox window manager, 29
- block device file type codes, 75
- blowfish encryption, 224
- boot directive, 12
- /boot/grub/grub.conf file, 10
- /boot/grub/menu.lst file, 10
- boot loaders, 2
 - available, 8–9
 - GRUB, 8–11
 - role, 7–8
- boot.log file, 170
- boot options
 - installation process, 4, 6
 - setting, 12–13
- /boot partition, 246
- boot problems, 14
 - dmesg messages, 16–17
 - GRUB kernel options, 14–15
 - rescue discs for, 15
 - root passwords, 16
- BOOTPROTO variable, 369

- broadband routers, 367, 367
- broadcast queries, 360
- browsing in IPP, 418
- BRU package, 326
- BSD (Berkeley Standard Distribution)
 - disk labels, 242
 - printing systems, 416, 424, 427
 - ps options, 110–111
- buffers, ring, 16–17
- Bugs section in man pages, 93
- build numbers in RPM, 299
- BusID setting, 24
- bzImage kernel, 13
- bzip2 tool, 314–315

C

- caches, ARP, 376
- cancel command, 427
- canonical name (CNAME)
 - records, 445
- case changes in Vi, 136
- case-sensitivity of usernames, 193
- cat program, 34, 67–68
- cd command, 57
- cdrecord program, 329, 337–338
- cdrtools package, 337
- CERT (Computer Emergency Response Team), 514
- CGI (Common Gateway Interface)
 - scripts, 490–491
- chage command, 203–204
- chain loaders, 12
- chains, iptables, 399–401, 399
- changing
 - passwords, 221
 - runlevels, 160–162
 - text case, 53
- character devices file type codes, 75

- checksums
 - CRC, 244
 - file database, 295
 - generating, 543–544
 - packet manager, 544–545
 - Tripwire, 542–543
- chgrp command, 79, 194, 219
- child processes, 109
- chipsets for video card, 18–19, 18
- chkconfig tool, 151
- chkrootkit program, 545
- chmod command, 78–81
- choosers, 412
- chown command, 78–79, 194, 202
- chroot jails, 485
- CHS translation schemes, 10
- CIDR (Classless Inter-Domain Routing), 361
- cifs filesystem, 272–273
- classes for IP addresses, 361
- Classless Inter-Domain Routing (CIDR), 361
- clients
 - DHCP, 368–369
 - FTP, 486–488
 - iptables traffic, 404
 - NTP, 450, 454
 - vs. servers, 365
 - VNC, 498–499
 - Web, 495–496
 - X, 17
- CNAME (canonical name) records, 445
- colons (:)
 - account configuration files, 204–205
 - chown, 79
 - directories, 55
 - group configuration files, 215
 - hardware addresses, 359
 - IP addresses, 360
 - scp, 407–408
 - Vi, 133
 - VNC clients, 498
- color depths in X configuration, 25
- color inkjet printers, 417
- combining text files, 67–68
- command-line interface (CLI). *See* shells
- command lines, generating, 70–71
- Command mode in Vi, 133
- command prompts for shells, 49
- commas (,)
 - cron jobs, 126
 - DNS servers, 440, 446
 - hostnames, 534
 - log files, 164
 - mount options, 275
 - symbolic permissions, 81
- comments
 - account configuration files, 205
 - aliases, 460
 - Apache, 489
 - /etc/fstab, 275
 - /etc/named.conf, 442
 - /etc/services, 396
 - inetd.conf, 154
 - log files, 164, 166
 - Samba, 477
 - shell scripts, 90
 - X configuration, 21
- Common Gateway Interface (CGI)
 - scripts, 490–491
- Common Unix Printing System (CUPS)
 - utility, 414, 417
 - configuration files, 417–419
 - printer definitions, 420
 - Web-based utilities, 420–422, 421
- compatible hardware, 30–31
- compiling source code, 318–321
- complaints as intrusion symptom, 538

- compress option, 168
- compress tool, 314–315
- compression options
 - log files, 168
 - tarballs, 314–315
- compromised passwords, 221–222
- Computer Emergency Response Team (CERT), 514
- Conectiva distribution, 3
- conflicts in packages, 322–324
- connections
 - MySQL, 501–502
 - TCP/IP, 356
 - testing, 377
- consoles, X configuration, 29–30
- content of files, 34, 66–67
- contexts in SELinux, 529–530
- copying files, 58–59
- corrupted log files, 538
- Courier e-mail servers, 462
- cp command, 58–59
- cpio program, 326–329, 335
- CPUs and processes
 - priority, 113
 - restrictions on, 116–117
 - time use, 112–113, 116
- Crack program, 221
- crackers, 219
- crashes as intrusion symptom, 538
- create option for log files, 168
- cron file, 170
- cron jobs, 125
 - log files, 166
 - system, 126–127
 - user, 127–128
- crontab utility, 127–128
- crontabs, 127
- cross-platform optical discs, 338–339

- CUPS (Common Unix Printing System)
 - utility, 414, 417
 - configuration files, 417–419
 - printer definitions, 420
 - Web-based utilities, 420–422, 421
- CUPS Driver Development Kit (DDK), 420
- cupsd daemon, 417
- cupsdisable command, 427
- cupsenable command, 427
- curl command, 496
- curly braces ({})
 - environment variables, 86
 - /etc/dhcpd.conf, 440
 - log files, 167
- current directories, 56–58
- custom.conf file, 29, 413
- custom startup files, 158–159
- cylinders, 10
- Cyrus IMAP servers, 461

D

- daemons, 125, 148
- daily option for log files, 168
- dashes (-) in cron jobs, 126
- data overrun protection for
 - partitions, 246
- data structures in filesystems, 74
- databases
 - installed file, 295–296
 - network accounts, 207
- date command, 449
- day setting in job scheduling, 128
- Days before change required field, 206
- Days between expiration and
 - deactivation field, 206
- Days until change allowed field, 206

- Days warning before password expiration field, 206
- dd utility, 261
- DDK (Driver Development Kit), 420
- DDoS (distributed denial-of-service) attacks, 518
- Debian distribution, 3
- Debian packages, 307
 - APT tool, 311–313, 314
 - conventions, 307–308
 - dpkg commands, 308–311
 - tools, 294
- debug level logs, 164
- Default OS option, 11
- default permissions, 82–83
- default routes, 373
- default shells, 189, 205
- DefaultDepth setting, 25
- definitions, glossary of, 556–591
- deleting
 - files, 60
 - groups, 216
 - print queue jobs, 426–427
 - shell command text, 53
 - user accounts, 207–208
- denial-of-service (DoS) attacks, 518
- dependencies, package, 296, 322–324
- Description section in man pages, 92
- desktop environment, 17, 29
- /dev directory, 11, 71
- /dev/fd devices, 241
- /dev/hd devices, 240–241
- /dev/ht devices, 241
- /dev/nht devices, 241
- /dev/nst devices, 241
- /dev/null file, 71
- /dev/nvram file, 242
- /dev/random file, 71
- /dev/scd devices, 241
- /dev/sd devices, 240–241
- /dev/st devices, 241
- /dev/urandom file, 71
- /dev/zero file, 71
- Device field, 275
- device files, 71–72
- Device Manager, 18, 18
- df utility, 266, 273–274
- dhclient client, 368
- DHCP (Dynamic Host Configuration Protocol), 6, 359
 - configuration, 368–369
 - IP addresses with, 438–441
- dhcp.leases file, 440
- dhcp package, 439
- dhcp-server package, 439
- dhcpcd client, 368
- dhcpcd program, 439
- diff command, 536–537
- dig program, 364
- digital subscriber line (DSL), 351
- direct administrative logins, 191
- directories, 53
 - creating, 62
 - current, 56–58
 - file type codes, 75
 - home, 196–197
 - LDAP, 455, 525
 - permissions, 77
 - removing, 62–63
- dirs command, 57
- Disabled mode in SELinux, 530
- disabling unused accounts, 222–223
- Disallowed IP or network addresses
 - setting, 158
- disaster recovery, backups for, 335–336
- disks
 - for backups, 325–326
 - exam essentials, 283–284

- partitioning. *See* partitions
 - RAID. *See* Redundant Array of Independent Disks (RAID)
 - review questions, 285–291
 - storage hardware overview, 238–242
 - summary, 282–283
 - DISPLAY environment variable, 88
 - DISPLAYMANAGER variable, 27
 - distributed denial-of-service (DDoS)
 - attacks, 518
 - distribution selection, 2–3
 - distribution-specific tools, 21
 - dm-crypt program, 515
 - dmesg file, 170
 - dmesg utility
 - messages, 16–17
 - video card chipsets, 19
 - DNS. *See* Domain Name System (DNS)
 - documentation
 - man pages, 91–94
 - programs, 95–96
 - video card chipsets, 18
 - dollar signs (\$)
 - environment variables, 85, 89
 - shells, 49
 - domain controller computers, 478
 - Domain Name System (DNS)
 - hostname resolution, 363–364, 441
 - BIND options, 442
 - domain configuration, 443–448
 - root zone configuration, 443
 - servers, 448
 - iptables traffic, 404
 - options, 371
 - Domain setting, 479
 - domains
 - configuring, 443–448
 - names, 362, 440
 - obtaining, 441
 - Samba, 478–479
 - SELinux, 529–530
 - DoS (denial-of-service) attacks, 518
 - dots (.)
 - chown, 79
 - directories, 57
 - forward zone files, 444–446
 - reverse zone files, 447
 - sysctl keys, 131
 - usernames, 193
 - dotted quad notation, 361
 - dpkg command, 308–311
 - drift file, 450
 - Driver Development Kit (DDK), 420
 - drivers
 - compiling, 320–321
 - kernel, 33–34
 - printer, 415, 420, 478
 - X configuration, 24
 - dselect utility, 311
 - DSL (digital subscriber line), 351
 - dual-core systems, 116
 - dump program, 328
 - /etc/fstab option, 275
 - for restoring, 329–331
 - duplicate files and features, dependency
 - issues from, 322
 - Dynamic Host Configuration Protocol (DHCP), 6, 359
 - configuration, 368–369
 - IP addresses with, 438–441
 - dynamic viewing of processes, 113–116, 114
-
- ## E
- e-mail, 455
 - aliases and forwarding, 460–461
 - protocols, 456
 - queues, 459–460
 - server selection, 461–462
 - SMTP servers, 456–458

- e2fsck tool, 258–259
- echo command, 393
- edit mode in Vi, 134
- editing files with Vi, 132
 - modes, 132–133
 - saving changes, 137
 - text editing, 134–136, 135
- EDITOR environment variable, 53, 88
- editors, invoking, 53
- edquota utility, 264–265
- effective user IDs, 123
- EFI (Extensible Firmware Interface)
 - specification, 8, 243
- ELILO boot loader, 245
- Emacs text editor, 51–52, 137
- Embedded Appliance Firewall, 394
- emerg priority logs, 164–165
- emergency recovery, 335
- Emulate3Buttons option, 22
- Enabled mode in SELinux, 530
- encryption
 - databases, 224
 - e-mail servers, 462
 - GPG, 528
 - passwords, 205–206, 220–221, 223
 - Samba, 475
 - security, 518–519
 - SSH, 405
 - wireless networks, 515
- ends of files, viewing, 68, 173–174
- env command, 86, 89
- envelopes, packets, 354
- environment variables, 84–85
 - common, 86–89
 - setting, 85–86
- equal signs (=)
 - Bash, 89
 - environment variables, 85–86
 - log files, 165
 - MySQL, 503
- errors
 - filesystem checking, 258–259
 - log files for, 172
 - partition protection from, 246
- /etc directory, 247
- /etc/aliases file, 460
- /etc/apache file, 489
- /etc/apt/sources.list file, 311
- /etc/cron.d directory, 125–126
- /etc/crontab file, 125–127
- /etc/cups directory, 417
- /etc/cups/cupsd.conf file, 418–419
- /etc/cups/ppd directory, 418
- /etc/cups/printers.conf file, 417–418
- /etc/dhcpd.conf file, 439–440
- /etc/event.d directory, 163
- /etc/exports file, 480–483
- /etc/fstab file, 259
 - filesystems, 274–276
 - logical volumes, 282
 - mounting devices, 267–268
 - quotas, 264
 - raid, 280
 - swap files, 261–263
 - unmounting devices, 271
- /etc/ftpusers file, 225, 485
- /etc/gdm file, 413
- /etc/group file, 193–195, 213–216
- /etc/gshadow file, 216
- /etc/host.conf file, 364
- /etc/hostname file, 380
- /etc/hosts.allow file, 157
- /etc/hosts.deny file, 157
- /etc/hosts file, 364, 443
- /etc/httpd directory, 489
- /etc/httpd/conf file, 489
- /etc/inetd.conf file, 153, 158
- /etc/inetd.d directory, 153
- /etc/init.d directory, 150–151
- /etc/init.d/mysqld script, 501

- /etc/init.d/ntpd restart script, 453
- /etc/inittab file, 27, 160, 162
- /etc/kde/kdm file, 412
- /etc/ldap.conf file, 525
- /etc/login.defs file, 197
- /etc/logrotate.conf file, **166–168**
- /etc/logrotate.d directory, 166–167
- /etc/mail/sendmail.cf file, 457
- /etc/modules file, 37
- /etc/modules.d directory, 37
- /etc/mtab file, 268
- /etc/mysql directory, 501
- /etc/named.conf file, 442–444, 448
- /etc/network/interfaces file, 369–370
- /etc/nsswitch.conf file, 364, 479, 520, 524–525
- /etc/ntp.conf file, 452–453
- /etc/pam.conf file, 521
- /etc/pam.d directory, 479, 521–522
- /etc/pam.d/login file, 521
- /etc/pam.d/passwd file, 223
- /etc/passwd file
 - account information, 520
 - contents, 190
 - creating, 223
 - editing, 194
 - groups, 195, 216
 - reviewing, 536
- /etc/portsentry directory, 541
- /etc/postfix/main.cf file, 458
- /etc/ppp/chap-secrets file, 375
- /etc/ppp/pap-secrets file, 375
- /etc/profile file, 83
- /etc/proftpd/proftpd.conf file, 484–485
- /etc/raidtab file, 278–280
- /etc/rc.d directory, 151, 416
- /etc/rc.d/boot.local file, 158
- /etc/rc.d/init.d directory, 150–151
- /etc/rc.d/rc.local file, 158
- /etc/resolv.conf file, 371
- /etc/runlevels directory, 151
- /etc/samba/lmhosts file, 476
- /etc/samba/smb.conf file, 475
- /etc/samba/smbpasswd file, 475–476
- /etc/securetty file, 226
- /etc/selinux/config file, 531
- /etc/services file, 382, 396, 532
- /etc/shadow file, 16
 - account information, 520
 - contents, 190
 - creating, 223
 - editing, 205–206
 - permissions, 207
 - reviewing, 536
- /etc/snort/snort.conf file, 539
- /etc/squid/squid.conf file, 494
- /etc/ssh directory, 408
- /etc/ssh/ssh_config file, 405, 410
- /etc/ssh/sshd_config file, 226, 405–406
- /etc/sudoers file, 192
- /etc/sysconfig directory, 27
- /etc/sysconfig/displaymanager file, 411
- /etc/sysconfig/network/routes file, 370
- /etc/sysconfig/network-scripts/ifcfg-eth0 file, 369–370
- /etc/sysconfig/sysctl file, 393
- /etc/sysctl.conf file, 132, 393
- /etc/syslog.conf file, 164–165, 171
- /etc/tripwire directory, 543
- /etc/vsftpd/vsftpd.conf file, 484, 486
- /etc/X11/gdm file, 413
- /etc/X11/gdm/gdm.conf file, 411
- /etc/X11/kdm file, 28, 411–412
- /etc/X11/xdm/Xaccess file, 412
- /etc/X11/xdm/xdm-config file, 28, 412
- /etc/xinetd.conf file, 155
- /etc/xinetd.d directory, 155
- /etc/yp.conf file, 523

- /etc/yum.conf file, 306
- /etc/yum.repos.d, directory, 306
- Ethereal packet sniffer, 541
- Ethernet networks, 351
- Ex mode in Vi, 133
- exclamation marks (!)
 - account passwords, 205
 - log files, 165
 - shell commands, 53
 - Vi files, 137
- executable form, 89
- execute permission, 75–77
- Exiting command for info pages, 95
- Expiration date field, 206
- expiring passwords, 221
- export command, 85–86
- exportfs command, 480–481
- exports
 - GPG keys, 527
 - NFS, 273, 480
- expressions
 - find, 63–64
 - regular, 67, 177
- ext2 (second extended filesystem), 248–249
- ext2fs filesystem, 248
- ext3fs (third extended filesystem), 249–250
- ext4fs (fourth extended filesystem), 249
- extended INT13 calls, 10
- extended partitions, 243
- Extensible Firmware Interface (EFI)
 - specification, 8, 243
- Extent Filesystem (XFS), 249–250

F

- FAQs (Frequently Asked Questions), 96
- Fast Filesystem (FFS), 249

- FAT (File Allocation Table) filesystem, 249–250
- FCEDIT environment variable, 53
- FDDI (Fiber Distributed Data Interface), 352
- fdformat utility, 241
- fdisk program, 250
 - partition creation, 252–254, 253
 - partition listing, 265–266
- Fedora distribution, 3
 - GRUB version, 10
 - Network Configuration tool, 374, 374
 - RPM for, 299–300
 - static IP addresses, 370
- Fetchmail program, 456
- FFS (Fast Filesystem), 249
- fg program, 51, 119
- Fiber Distributed Data Interface (FDDI), 352
- Fibre Channel networks, 352
- File Allocation Table (FAT) filesystem, 249–250
- file command, 78
- File Transfer Protocol (FTP)
 - access control, 225
 - clients, 486–488
 - network backups, 334
 - security issues, 483, 519
 - servers, 483
 - choosing, 483–484
 - configuring, 484–486
- filenames, 74
 - filename completion, 51–52
 - optical disks, 339
 - packages, 302
- files, 53
 - combining, 67–68
 - contents, 34, 66–67
 - copying, 58–59

- creating, 60
 - device, 71–72
 - editing with Vi. *See* Vi editor
 - links to, 60–61
 - listing, 54–56
 - locating, 63–66
 - log. *See* log files
 - moving and renaming, 59
 - ownership, 72–74, 79
 - permissions. *See* permissions
 - redirecting, 69–70
 - removing, 60
 - sharing with Samba, 476
 - startup, 158–159
 - type codes, 74–78
 - type information, 78
 - verifying, 537
 - viewing, 67–68
 - Files section
 - man pages, 92
 - X configuration, 22
 - Filesystem in Userspace (FUSE), 488
 - filesystems
 - ACL support, 83–84
 - creating, 256–257
 - data structures, 74
 - error checking, 258–259
 - meanings, 238
 - networks, 271–276
 - optical disks, 337
 - options, 248–249
 - partitions for, 246, 248–249, 256–259, 271–276
 - quotas, 263–265
 - standard, 274–276
 - filter table, 399–401
 - filters
 - Ghostsript, 416
 - packet, 393–394, 399–400, 399
 - proxy, 394
 - find command, 63–65
 - Firestarter tool, 395
 - firewalls, 393–394
 - iptables, 398–404, 399
 - purpose, 394–395, 394
 - rules, 400–403
 - server ports, 396–398
 - software, 395
 - Foomatic printer definitions, 420
 - forcing installations, 323
 - foreground processes, 119
 - forking processes, 109
 - FORWARD chain, 399, 399, 401
 - forward zone files, 444–447
 - forwarding e-mail, 460–461
 - forwarding-only DNS servers, 442
 - fourth extended filesystem (ext4fs), 249
 - fragmented swap files, 262
 - frame buffer devices, 19
 - frames for packets, 354
 - Free Software Foundation (FSF), 95
 - free tool, 260–261
 - FreeRADIUS servers, 526
 - Frequently Asked Questions (FAQs), 95
 - Fresh RPMs repository, 307
 - fsck tool, 258–259
 - FSF (Free Software Foundation), 95
 - FTP. *See* File Transfer Protocol (FTP)
 - full-duplex transmissions, 353
 - full recursive DNS lookups, 442
 - FUSE (Filesystem in Userspace), 488
 - fuseftp client, 488
 - fvwm window manager, 29
-
- G**
- gateways
 - Fedora, 370
 - TCP/IP, 356
 - gawk tool, 175

- GCC (GNU Compiler Collection), 318
- gdisk program, 252, 266
- GDM (GNOME Display Manager), 26
 - configuring, 28–29
 - network connections, 411
- gdm.conf file, 28, 413
- gdm script, 27
- gedit text editor, 137
- general-purpose system statistics, 120–121
- Gentoo distribution, 3
 - GRUB version, 10
 - recompilation, 298
 - runlevels, 150–151
 - startup scripts, 151
 - XDMCP, 27
- getfacl command, 84
- gFTP tool, 488
- Ghostscript language, 415–416
- Ghostscript Web page, 417
- GIDs (group IDs), 72–73
 - group configuration files, 215
 - mapping to groups, 194–196
 - user accounts, 189, 205
- gigabit Ethernet networks, 351
- global replacement in Vi, 136
- GNOME (GNU Network Object Model Environment) desktop environment, 29
- GNOME Display Manager (GDM), 26
 - configuring, 28–29
 - network connections, 411
- GNOME Partition Editor, 251
- gnome-system-monitor tool, 114
- GNOME Toaster tool, 337
- GNU
 - awk and gawk, 175
 - Ghostscript, 415
 - Hurd kernel, 12
 - Parted tool, 250, 254–256
 - ps options, 110–111
- GNU Compiler Collection (GCC), 318
- GNU Network Object Model
 - Environment (GNOME) desktop environment, 29
- gpasswd command, 214–215, 217
- GPG (GNU Privacy Guard), 519, 526
 - encrypting and decrypting data, 528
 - keys, 527–528
 - signatures, 529
- gpg program, 527–529
- GProFTPd FTP servers, 483, 485
- GPT (GUID partition table) partitioning system, 242–244
- Grand Unified Boot Loader (GRUB), 8–9
 - configuring, 9–10
 - global options, 10–11
 - kernel additions to, 13
 - kernel options, 14–15
 - OS boot options, 12–13
 - partitions for, 244–245
- greater-than symbols (>)
 - file combining, 67
 - redirection, 69
 - shells, 49
- grep command, 66–67, 174–176
- group administrators, 214
- group IDs (GIDs), 72–73
 - group configuration files, 215
 - mapping to groups, 194–196
 - user accounts, 189, 205
- groupadd command, 212–213
- groupdel command, 216
- groupmod command, 214
- groups, 72
 - adding, 212–213
 - deleting, 216

- mapping GIDs to, 194–196
- modifying, 213–216
- multiple, 218–219
- overview, 193–194
- owners, 72
- project, 217–218
- user private, 217
- volume, 280–282
- growisofs program, 337–338
- grpconv program, 223
- GRUB. *See* Grand Unified Boot
 - Loader (GRUB)
- grub.conf file, 13–14
- grub-install program, 9, 13
- grub utility, 12
- Guarddog tool, 395
- GUI environments
 - firewall configuration tools, 395
 - installers, 5
 - network configuration tools,
 - 373–374, 374
 - remote logins, 412–413
 - text editors, 137
 - X configuration logins, 26, 26
- GUID partition table (GPT) partitioning
 - system, 242–244
- guides, 96
- Gutenprint drivers, 420
- gzip tool, 314–315

H

- hackers, 219
- half-duplex transmissions, 353
- hard disks. *See* disks
- hard links, 60–61, 73
- hardware, 30
 - backups, 324–326
 - compatible, 30–31

- identifying, 31–34
- kernel modules, 35–37
- network configuration, 366
- networks, 350–354, 353
- storage, 238
 - configuration, 240–242
 - device types, 239–240
 - disks. *See* disks
- hardware addresses, 359
- Hardware Compatibility List (HCL), 30
- hardware RAID, 278
- hash marks (#)
 - aliases, 460
 - Apache, 489
 - /etc/fstab, 275
 - /etc/services, 396
 - inetd.conf, 154
 - log files, 164, 166
 - shell scripts, 90
 - shells, 49
 - X configuration, 21
- hashes
 - password, 220, 223
 - Tripwire, 542–543
- HCL (Hardware Compatibility List), 30
- head program, 173
- headers
 - in compilation, 318–319
 - GPT, 244
- heavy loads, verifying, 171
- help resources, 91
 - info pages, 94–95
 - Internet-based, 96–97
 - man pages, 91–94
 - program documentation, 95–96
- HFS (Hierarchical Filesystem), 249, 339
- High-Performance Filesystem (HPFS),
 - 249–250

High-Performance Parallel Interface (HIPPI), 352
 historical login data, 211–212
 history command, 53
 history in shells, 52–53
 History section in man pages, 93
 home directories
 overview, 196–197
 tildes for, 57
 user accounts, 189, 205
 HOME environment variable, 87
 /home partition, 246
 HorizSync setting in X configuration, 23
 host program, 363–364, 379
 hosting, virtual, 492–493
 hostname command, 379–380
 HOSTNAME environment variable, 86
 hostnames, 362–364
 DNS for, 363–364, 441
 BIND options, 442
 domain configuration, 443–448
 root zone configuration, 443
 servers, 448
 listing, 379–380
 Samba servers, 476
 hot standby support, 277
 HOWTO documents, 96
 HPFS (High-Performance Filesystem), 249–250
 .htaccess files, 490
 HTTP (Hypertext Transfer Protocol), 489
 CUPS, 420
 security issues, 519
 httpd.conf file, 489–490
 httpd package, 489
 httpd2.conf file, 489
 HTTPS protocol, 519
 hubs, 352, 353

human element in security, 519
 hung processes, 116
 Hypertext Transfer Protocol (HTTP), 489
 CUPS, 420
 security issues, 519

I
 ICANN (Internet Corporation for Assigned Names and Numbers), 441
 IceWM window manager, 29
 ICMP (Internet Control Message Protocol), 356
 identity verification, 212
 IDS (intrusion detection system), 539–541, 540
 IETF (Internet Engineering Task Force), 356
 iface command, 369
 ifconfig command, 359, 372–373, 376
 ifdown command, 372
 ifup command, 372
 IMAP (Internet Message Access Protocol), 456, 461–462
 importing GPG keys, 527–528
 in-addr.arpa domain, 447
 in.ftpd server, 153
 in.telnetd server, 405
 incompatible libraries, dependency issues from, 322
 incremental backups, 334–336
 inetd.d directory, 153–155
 inetd program, 153–155, 405
 info pages, 91, 94–95
 init program, 160–162
 initial RAM disks, 321
 inkjet printers, 417
 inodes, 73–74, 264

- INPUT chain, 399, 399
 - input/output
 - redirection, 68
 - system statistics, 121–122
 - insmod program, 35
 - installations, 2
 - boot method, 4
 - distribution selection, 2–3
 - installer interaction, 5
 - media, 4–5
 - performing, 5–7
 - installed file database, 295–296
 - inted.conf file, 153–155
 - Internet-based help resources, 96–97
 - Internet Control Message Protocol (ICMP), 356
 - Internet Corporation for Assigned Names and Numbers (ICANN), 441
 - Internet Engineering Task Force (IETF), 356
 - Internet Message Access Protocol (IMAP), 456, 461–462
 - Internet Packet Exchange (IPX) protocol, 357
 - Internet Printing Protocol (IPP), 418
 - Internet service providers (ISPs) for time sources, 451
 - internets, 356
 - intrusion detection, 537
 - checksum generation, 543–544
 - ckrootkit, 545
 - intrusion symptoms, 537–538
 - log files, 172, 545–546
 - packet managers, 544–545
 - PortSentry, 540–541
 - Snort, 538–540, 540
 - Tripwire, 542–543
 - Wireshark, 541–542, 542
 - intrusion detection system (IDS), 539–541, 540
 - invoking editors, 53
 - iostat utility, 121–122
 - IP addresses
 - with DHCP, 438–441
 - overview, 359–362
 - static, 369–373
 - ipchains tool, 395, 398
 - ipfwadm tool, 395
 - IPP (Internet Printing Protocol), 418
 - iptables tool, 395, 398
 - configuring, 403–404
 - firewall rules, 400–403
 - packet filter architecture, 399–400, 399
 - IPv6, 358
 - IPX (Internet Packet Exchange) protocol, 357
 - IPX/SPX protocol, 357
 - ISO-9660 filesystem, 249
 - isofs module, 34
 - ISPs (Internet service providers) for time sources, 451
 - iwconfig command, 366, 368
-
- ## J
- JFS (Journaled Filesystem), 249
 - job numbers, 119
 - jobs
 - print queue, 426–427
 - process, 119
 - scheduling, 125–129
 - Joliet filesystem, 249, 339
 - Journaled Filesystem (JFS), 249
 - journaling filesystems, 249–250, 259

K

K Desktop Environment (KDE), 29, 307
 K3B tool, 337
 KDCs (Key Distribution Centers), 455
 KDE (K Desktop Environment), 29, 307
 KDM (KDE Display Manager), 26
 configuring, 28
 network connections, 411
 kdm script, 27
 kdmrc file, 413
 KEdit text editor, 137
 Kerberos protocol, 455
 kernel option, 12
 kernel ring buffers, 16–17
 kernels, 2
 adding, 13
 compiling, 320–321
 drivers, 33–34
 GRUB options, 14–15
 modules, 33, 35–37
 as process, 108
 system control data, 131–133
 upgrading, 303
 version information, 129–131
 Kernighan, Brian W., 175
 Key Distribution Centers (KDCs), 455
 keyboards
 as standard input device, 68
 X configuration, 22
 keyrings in GPG, 527
 keys
 GPG, 527–528
 SSH, 408–409
 system control, 131–132
 kill command, 117–118, 159
 killall command, 118, 159
 killing processes, 117–118, 159
 klogd daemon, 163

Knoppix package, 15
 Konqueror program, 425, 426
 kpm tool, 114
 ksysv utility, 152, 152
 kyum tool, 306

L

language options in installation, 6
 laptop computer security issues, 515
 LaserWriter printer drivers, 478
 last command, 211–212
 Last password change field, 206
 Last topic command for info pages, 95
 lastlog file, 170
 launching shell programs, 50–51
 LBA (linear block addressing)
 mode, 10
 LD_LIBRARY_PATH environment
 variable, 87
 LDAP (Lightweight Directory Access Protocol), 207, 455, 525
 LDP (Linux Documentation Project),
 96–97
 leases, DHCP, 368, 439
 less program, 17, 68, 173, 175
 less-than symbols for redirection, 68
 /lib/modules directory, 33
 libparted tools, 251
 libraries
 in compilation, 318–319
 dependency issues from, 322–323
 Lightweight Directory Access Protocol
 (LDAP), 207, 455, 525
 LILO (Linux Loader), 8–9, 244–245
 line mode in Vi, 134
 linear block addressing (LBA)
 mode, 10
 linear RAID approach, 276–277

links

- creating, 60–61
- file type codes, 75
- inodes for, 73
- optical disks, 338
- permissions, 77
- Linux Documentation Project (LDP), 96–97
- Linux Loader (LILO), 8–9, 244–245
- listing files, 54–56
- Livna repository, 307
- lmhosts file, 476
- ln command, 60–61
- loading
 - Apache modules, 490
 - kernel modules, 35–36
- LOADLIN boot loader, 9
- loads
 - averages, 116
 - verifying, 171
- local network tools, 376, 531–532
- local program bugs, 516–517
- local user complaints as intrusion symptom, 538
- LocalTalk protocol, 352
- locate command, 65
- locating files, 63–66
- log files, 163
 - analysis tools, 177
 - awk and sed tools, 175–177
 - common, 170–171
 - exam essentials, 178–179
 - for intrusion detection, 172, 538, 545–546
 - monitoring, 545–546
 - options, 164–166
 - for problem identification, 171–172
 - remote servers for, 169–170
 - review questions, 180–186
 - rotating, 125, 166–169

- scanning, 172–177
- for services, 148
- summary, 178
- syslogd for, 163–164
- logcheck.sh file, 177
- Logcheck tool, 177
- logical partitions, 243, 253
- logical volumes, 280, 282
- login program, 225–226
- logins
 - access control, 225
 - historical data, 211–212
 - privileges, 189
 - remote. *See* remote logins
 - switching identities after, 191–192
 - user account shells, 536
- logrotate tool, 166–168
- logtail utility, 177
- long text files, viewing, 68
- loop-AES program, 515
- loopback interface configuration, 371
- loopback option in optical disks, 338
- loopback traffic with iptables, 404
- low-level disk formats, 241
- lp program, 424–425
- lpc utility, 427
- lpd tool, 414, 423
- lpmove command, 427
- lpq utility, 426–427
- lpr program, 414, 424–425
- lprm command, 427
- lpstat utility, 427
- ls program, 51
 - options, 54–56
 - wildcards, 56
- lsmod command, 33–34
- lspci command, 19, 31
- lsusb command, 32
- lvcreate command, 282
- lvdisk command, 282

LVM tool, 280–281
 lvremove command, 282
 lvresize command, 282
 lvscan command, 282
 lynx program, 495

M

m4 utility, 458
 MAC (mandatory access control), 529
 MAC (Media Access Control)
 addresses, 359
 machine names, 362
 magnetic devices, 239, 241
 magnetic tape, 239–240
 mail. *See* e-mail
 mail exchanger (MX) records, 446
 mail options for log files, 168
 mail transfer agents (MTAs), 456
 mail user agents (MUAs), 456
 maillog file, 170
 mailq command, 459
 make utility, 318–321
 makewhatis command, 94
 man pages, 91–94
 mandatory access control (MAC), 529
 Mandrake distribution, 3
 Mandriva distribution, 3
 mangle table, 399
 mapping UIDs and GIDs to users and
 groups, 194–196
 masks
 DHCP, 440
 network, 360, 370
 user, 82–83
 master boot record (MBR), 8
 partitioning system, 242–243
 protective, 244
 MD4 encryption, 224
 MD5 (Message Digest 5) hash, 223–224
 md5sum program, 543–544
 mdadm package, 278
 Media Access Control (MAC)
 addresses, 359
 media for installation, 4–5
 /media partition, 247
 memory
 process use, 113
 statistics, 122
 swap space for, 259–263
 menu.lst file, 13–14
 menuconfig program, 319
 Message Digest 5 (MD5) hash, 223–224
 messages, dmesg, 16–17
 messages log file, 170
 Metacity window manager, 29
 midnight setting in job scheduling, 128
 mirroring, RAID, 277
 mismatched names, dependency issues
 from, 322–323
 missing libraries, dependency issues
 from, 322–323
 missing log entries, 172, 538, 546
 mixing case in passwords, 220
 mkdir command, 62
 mkdosfs utility, 257
 mke2fs tool, 256
 mkfs program, 256–257
 mkfs.ext2 program, 256
 mkinitramfs utility, 321
 mkinitrd utility, 321
 mkisofs program, 329, 337–338
 mknod command, 71–72
 mkraid command, 280
 mkreiserfs utility, 257
 mkswap tool, 261, 263
 /mnt partition, 247
 mode lines in X configuration, 23

- modes
 - FTP, 488
 - SELinux, 530
 - symbolic, 80–81
 - Vi, 132–133
- modinfo command, 34
- modprobe program, 35–36, 264, 366
- module stacks, 37
- modules
 - Apache servers, 490
 - kernel, 33, 35–37
 - PAM, 223, 455, 521–523
- monitor options in X configuration, 23
- monitoring
 - log files, 545–546
 - queues, 424–427, 426
- monthly option for log files, 168
- more program, 68
- mount command, 264, 266–270, 272–273
- Mount options field, 275
- Mount point field, 275
- mount points, 245–247, 266
- mounting partitions, 266–270
- mouse, X configuration, 22
- moving
 - files, 59
 - shell text, 53
- Moving up command for info pages, 94
- mt program, 240, 331–333
- MTAs (mail transfer agents), 456
- MUAs (mail user agents), 456
- multiple disks, partitions for, 245
- multiple groups for users, 218–219
- multiple partitions, 248
- multitasking systems, 190–191
- multiuser concepts, 188
 - groups, 193–194
 - home directories, 196–197

- mapping UIDs and GIDs to users and groups, 194–196
 - user accounts, 188–193
- mv command, 59
- MX (mail exchanger) records, 446
- my.cnf file, 500
- MySQL language
 - configuring, 500–501
 - connections, 501–502
 - SQL packages, 499–500
- mysqladmin command, 501
- mysqld daemon, 501

N

- Name section in man pages, 92
- name server (NS) records, 445–446
- name servers
 - DHCP, 440
 - troubleshooting, 379–380
- Name Service Switch (NSS), 520
- named file, 441
- names
 - dependency issues from, 322–323
 - groups, 215
 - hostnames. *See* hostnames
 - kernel files, 13
 - packages, 302
 - RPM package files, 299
- NAT (network address translation), 362, 399
- nat table, 399
- NEdit text editor, 137
- Neighbor Discovery Protocol (NDP), 360
- Nessus scanner, 532–535, 534–535
- Netatalk package, 357
- NetBEUI protocol, 357
- NetBIOS protocol, 357, 440

- netstat tool, 378–379, 531–532
- network activity
 - as intrusion symptom, 537
 - tools, 531–532
- network address translation (NAT), 362, 399
- network addresses
 - hardware, 359
 - hostnames, 362–364
 - IP, 359–362
 - overview, 358–359
- network-admin (Network Settings) tool, 373
- Network Configuration tool, 373
- Network File System (NFS), 382, 480
 - exports, 273
 - network backups, 334
 - servers, 480
 - configuring, 480–482
 - security, 482–483
- Network Information Service (NIS), 196, 207, 455, 523–524
- network installations, 4–5
- network masks
 - description, 360
 - DHCP, 440
 - Fedora, 370
- network servers, 438
 - authentication, 454–455, 523–526
 - e-mail. *See* e-mail
 - exam essentials, 463, 503–504
 - FTP, 483–488
 - hostnames, 441–448
 - IP addresses, 438–441
 - MySQL, 499–503
 - NFS, 480–483
 - review questions, 464–471, 505–511
 - Samba. *See* Samba servers
 - summary, 462, 503

- time, 448–454, 450, 453
- Web. *See* Web servers
- Network Settings tool
 - (network-admin), 373
- networks
 - account databases, 207
 - ARP cache, 376
 - authentication, 523–526
 - backups, 333–334
 - bind option, 157
 - configuration overview, 365–366
 - connectivity testing, 377
 - DHCP configuration, 368–369
 - exam essentials, 382–383, 428–429
 - filesystems, 271–276
 - firewalls. *See* firewalls
 - general tools, 380–382
 - GUI configuration tools, 373–374, 374
 - hardware, 350–354, 353
 - hardware configuration, 366
 - in installation process, 6
 - name server troubleshooting, 379–380
 - packets, 353–354
 - ports, 364–365
 - PPP, 375
 - printing. *See* printing
 - protocol stacks, 354–358, 355
 - remote logins. *See* remote logins
 - repositories, 296–297
 - review questions, 384–390, 429–436
 - route tracing, 377–378
 - routing between, 392–393
 - static IP address configuration, 369–373
 - status, 378–379
 - summary, 382, 428

- Windows remote access tools, 496–499
- wireless options, 366–368, 367
- New Technology Filesystem (NTFS), 249
- newaliases command, 460
- newgrp command, 194, 214–215, 218–219
- newrole command, 530
- Next page command for info pages, 94
- NFS. *See* Network File System (NFS)
- nfsserver script, 480
- nice commands, 116–117
- NIS (Network Information Service), 196, 207, 455, 523–524
- NIS+ protocol, 523
- NIS YP and Switch (NYS) protocol, 523
- nmap command, 533
- Nmap scanner, 532–533
- nmbd server, 475
- NNTPSERVER environment variable, 87
- no_access option, 158
- nodes in info pages, 94
- nonvolatile RAM (NVRAM), 239
- noon setting in job scheduling, 128
- now setting in job scheduling, 128
- NS (name server) records, 445–446
- nslookup program, 363, 379
- NSS (Name Service Switch), 520
- nss_ldap package, 525
- NTFS (New Technology Filesystem), 249
- NTLDR boot loader, 9
- ntp package, 452
- NTP protocol
 - basics, 449–450, 450
 - clients, 454
 - overview, 448–449

- server configuration, 452–454, 453
- time sources, 451
- ntpd package, 452
- ntpdate program, 454
- ntpq program, 453, 453
- ntsysv utility, 152
- numbers in passwords, 220
- NVRAM (nonvolatile RAM), 239, 242
- NVRAM Wakeup utility, 242
- NYS (NIS YP and Switch) protocol, 523

O

- off-site complaints as intrusion symptom, 538
- oftpd FTP servers, 484
- 1024-cylinder limit, 10
- only_from option, 158
- open mail relays, 457
- open ports, 531–535
- open single quote marks (')
 - sendmail configuration, 458
 - text within, 70–71
- Open System Interconnection (OSI) model, 355, 355
- OpenPrinting Database, 30
- OpenPrinting Web page, 417
- OpenSUSE network connections, 411
- /opt partition, 247
- optical media, 4–5
 - for backups, 325–326, 329
 - description, 239
 - working with, 241
 - writing to, 336–339
- optional partitions, 245–248
- Options section in man pages, 92
- or netmasks, 360
- order reversal in passwords, 220

- OS
 - boot options, 12–13
 - kernel additions to, 13
- OS Loader, 9
- OSI (Open System Interconnection)
 - model, 355, 355
- OUTPUT chain, 399, 399
- output redirection, 68
- overflow protection for partitions, 246
- ownership, 72
 - files, 72–74, 79
 - groups, 73

P

- packages
 - Debian, 307–313
 - dependencies and conflicts, 322–324
 - exam essentials, 340
 - file collections, 294–295
 - filenames, 302
 - in installation process, 6
 - installed file database, 295–296
 - network repositories, 296–297
 - overview, 294
 - rebuilding, 297–298, 323
 - review questions, 341–347
 - RPM, 298
 - distributions and conventions, 299–300
 - rpm commands, 300–303
 - Yum, 303–307, 306
 - source code compilation, 318–321
 - summary, 339
 - tarballs, 314–318
 - verifying, 537
- packet filters
 - architecture, 399–400, 399
 - firewalls, 393–394
 - packet managers, 544–545
 - packet sniffer programs, 538–542, 540, 542
 - packets, network, 353–354
 - PAGER environment variable, 87
 - PAM (Pluggable Authentication Modules)
 - configuring, 521–523
 - LDAP support, 455
 - passwd, 223
 - pam_ldap package, 525
 - pam_unix.so file, 224
 - parallel ATA (PATA) drives, 240
 - Parallel Line Interface Protocol (PLIP), 350
 - parameters for shells, 50
 - parent process IDs (PPIDs), 112
 - parent processes, 109, 109
 - Parted tool, 197, 250, 254–256
 - partial restores, 335
 - partition tables, 244
 - partitions, 241–242
 - creating, 251–256, 253
 - exam essentials, 283–284
 - filesystems, 248–249, 256–259
 - GPT, 242–244
 - identifying, 265–266
 - in installation process, 6
 - MBR, 243
 - mounting, 266–270
 - network filesystems, 271–276
 - optional, 245–248
 - quotas, 263–265
 - RAID, 276–280
 - requirements, 244–245
 - review questions, 285–291
 - root, 12
 - summary, 282–283
 - swap space, 259–263

- tools, 250–251
- unmounting, 270–271
- passive mode in FTP, 488
- passwd utility, 16, 200–201
- Password-cracking programs, 221
- passwords
 - BIOS, 515
 - compromised, 221–222
 - enforcing, 219–221
 - group configuration files, 215
 - groups, 214–215
 - MySQL, 501, 503
 - NIS, 524
 - protecting, 189
 - root, 16
 - Samba, 475
 - setting, 200–204
 - shadow, 190, 221–224
 - SMB/CIFS, 424
 - SSH, 405
 - stolen, 516
 - system accounts with, 536
 - user account configuration files, 204–206
 - wireless networks, 368
- PATA (parallel ATA) drives, 240
- PATH environment variable, 50, 87–88
- payloads in packets, 354
- PCI (Peripheral Component Interconnect) standard, 31–32
- PCL (Printer Control Language), 415
- pdisk tool, 252
- periods (.). *See* dots (.)
- Peripheral Component Interconnect (PCI) standard, 31–32
- permission bits, 76
- permissions, 72–73
 - ACLs, 83–84
 - components, 72–74
 - default, 82–83
 - modifying, 78–82
 - processes, 123–125
 - Samba, 477
 - type codes, 74–78
 - user accounts, 72, 189
- Permissive mode in SELinux, 530
- PGP (Pretty Good Privacy), 526
- phishing, 519
- PHP: Hypertext Preprocessor (PHP)
 - scripts, 490–492
- php.ini file, 492
- physical access in security, 515
- physical volumes, 281
- PIDs (process IDs), 112
- ping tool
 - connectivity testing, 377
 - NTP servers, 451
- pipes
 - file type codes, 75
 - working with, 69–70
- plain-text transfer mode in FTP, 488
- PLIP (Parallel Line Interface Protocol), 350
- Pluggable Authentication Modules (PAM)
 - configuring, 521–523
 - LDAP support, 455
 - passwd, 223
- plus signs (+)
 - job scheduling, 128
 - NTP servers, 453
- Point-to-Point Protocol (PPP), 351, 375
- pointer (PTR) records, 446
- policies
 - iptables, 400–401
 - printing, 419
 - SELinux packages, 530
- pool.ntp.org domain, 451
- POP (Post Office Protocol), 456, 461–462

- popd command, 57
- port forwarding, 406
- ports
 - firewalls, 396–398
 - FTP, 488
 - network, 364–365
 - open, 531–535, 534–535
 - TCP/IP, 358
- portsentry.conf file, 541
- PortSentry IDS, 540–541
- portsentry package, 541
- Post Office Protocol (POP), 456, 461–462
- postfix program, 459
- PostgreSQL package, 500
- postrotate option, log files, 168–169
- PostScript language
 - overview, 415
 - with Samba, 478
- PostScript Printer Definition (PPD)
 - files, 418
- PPIDs (parent process IDs), 112
- PPP (Point-to-Point Protocol), 351, 375
- ppp-off command, 375
- ppp-on command, 375
- ppp-on-dialer command, 375
- PPP over Ethernet (PPPoE), 351
- precompiled packages, 297
- prerotate option for log files, 168–169
- Pretty Good Privacy (PGP), 526
- Previous page command for info
 - pages, 94
- primary boot loaders, 8
- primary GIDs, 205
- primary partitions, 243, 253
- principles in SELinux, 529
- Printer Control Language (PCL), 415
- printer drivers, 415
- PRINTER environment variable, 88
- printing, 414
 - architecture, 414
 - CUPS configuration, 417–422, 421
 - Ghostscript, 415–416
 - to network printers, 423–424
 - PostScript, 415
 - print queues, 414, 424–427, 426
 - printer selection, 417
 - printer sharing, 478
 - systems, 416–417
- priorities
 - log files, 164–166
 - processes, 113, 116–117
- private groups, 217
- private keys
 - GPG, 527
 - SSH, 408
- privileged ports, 398
- privileges for user accounts, 189
- /proc filesystem, 34
- /proc/sys directory, 131
- /proc/version file, 131
- process IDs (PIDs), 112
- processes
 - CPU use restrictions, 116–117
 - exam essentials, 138
 - examining, 109–116, 114
 - foreground and background, 119
 - job scheduling, 125–129
 - kernel information, 129–133
 - killing, 117–118
 - overview, 108–109, 109
 - permissions, 123–125
 - review questions, 139–145
 - runlevels, 159–162
 - summary, 137–138
 - system statistics, 119–122
- ProFTPD FTP servers, 483
- program prompts, 192

- program-specific files, 190
- programs
 - behavior changes as intrusion symptom, 538
 - bugs, 516–517
 - documentation, 95–96
 - launching, 50–51
- project groups, 217–218
- prompts, 49, 192
- protective MBR, 244
- protocol stacks
 - OSI model, 355, 355
 - overview, 354–355
 - TCP/IP, 356
 - TCP/IP alternatives, 356–358
- provision information for packages, 296
- proxy filters, 394
- proxy servers, 494–495, 495
- ps command
 - options, 110–111
 - output, 111–113
 - process searches, 416
 - process status, 109
- PS_PERSONALITY environment variable, 110
- PS1 environment variable, 87
- PS2 environment variable, 87
- PTR (pointer) records, 446
- public keys
 - GPG, 527
 - SSH, 408
- public NTP servers, 451
- pull protocols, 456
- pump client, 368
- punctuation in passwords, 220
- PureFTPD servers, 484
- push protocol, 456
- pushd command, 57
- pvcreate command, 281–282
- pvdisplay command, 281

- pvextend command, 282
- pvresize command, 281
- pvsplit command, 281
- pwconv program, 223
- pwd command, 56
- PWD environment variable, 87

Q

- QMS magicolor printer drivers, 478
- Qpopper server, 462
- QTParted program, 251
- question marks (?) for wildcards, 56
- queues
 - e-mail, 459–460
 - monitoring and controlling, 424–427, 426
 - print, 414
- quota package, 263
- quota v1 support, 263
- quota v2 support, 263
- quotacheck command, 265
- quotas, disk, 263–265

R

- RADIUS authentication, 525–526
- RAID. *See* Redundant Array of Independent Disks (RAID)
- raidtools tool, 278
- random-access devices, 324
- random numbers, 71
- rc-update tool, 151
- rdesktop program, 496–497
- read permission, 75–77
- README files, 95–96
- RealVNC site, 498
- rebuilding packages, 297–298, 323
- recovery, backups for, 335–336

- Red Hat distribution, 3
 - GRUB version, 10
 - RPM for, 299
- redirecting files, 69–70
- Redundant Array of Independent Disks (RAID), 245, 276
 - benefits, 276
 - configuring, 278–280
 - designing, 277–278
 - forms, 276–277
- refresh rate in X configuration, 23
- regular expressions, 67, 177
- ReiserFS filesystem, 249–250
- relative directory names, 57
- release numbers for RPM, 299
- reliability, RAID for, 276
- reload command, 150, 155
- remote access tools for Windows, 496–499
- remote logins, 405–413
 - GUI, 412–413
 - remote access server setup, 405–406
 - secure protocols, 222
 - SSH keys, 408–409
 - text-mode, 406–408
 - X programs, 409–411
- remote network scanners, 532–535, 534–535
- remote servers
 - for log files, 169–170
 - setup, 405–406
- removable solid-state storage, 239, 241, 325
- removing
 - directories, 62–63
 - files, 60
 - kernel modules, 37
 - print queue jobs, 426–427
- renaming files, 59
- renice commands, 116–117
- repeated login failures, 545
- replacing
 - packages, 323
 - Vi text, 136
- repositories, network, 296–297
- repquota command, 265
- Requests for Comments (RFCs), 356
- rescue discs, 15
- resetting root passwords, 16
- resolution in X configuration, 23, 25
- resolving hostnames, 363–364
- restart command, 150, 155
- restoring backups, 329–333
- reverse zone files, 444, 447–448
- RFCs (Requests for Comments), 356
- ring buffers, 16–17
- rm command, 60
- rmdir command, 62–63
- rmmod command, 37
- rndc utility, 448
- Rock Ridge
 - cross-platform discs, 338–339
 - ISO-9960 support, 250, 270
 - optical disk support, 337
- root
 - access control, 225–226
 - log file ownership, 173
 - passwords, 16, 516
 - permissions, 77
 - security for, 192
 - UIDs, 536
- /root directory, 245
- root DNS servers, 442
- root kits, 545
- root partitions, 12, 245
- root window, 29
- root zones, 443
- rooted systems, 517
- rootnoverify option, 12
- rotating log files, 166–169
- route command, 372, 392–393
- route tracing, 377–378

- routers
 - broadband, 367, 367
 - DHCP, 440
 - TCP/IP, 356
 - routing between networks, 392–393
 - RPM Package Manager (RPM), 3, 294, 544–545
 - RPM packages, 298
 - distributions and conventions, 299–300
 - rpm commands, 300–303
 - Yum, 303–307, 306
 - rpmbuild program, 297, 319
 - RPMFind site, 307
 - rsync tool, 333–334
 - rules
 - firewalls, 400–403
 - Snort, 539
 - runlevel command, 160
 - runlevels, 159
 - changing, 160–162
 - role, 159–160
 - services, 149–152, 152
 - XDMCP server, 27
-
- S**
- Samba package, 272
 - Samba servers, 474–475
 - file sharing, 477
 - hostname resolution, 476
 - options, 475–476
 - printer sharing, 478
 - Windows domains with, 478–479
 - Samba Web Administration Tool (SWAT), 226
 - SANE (Scanner Access Now Easy)
 - project, 30
 - sar utility, 120–121
 - SAS (Serial Attached SCSI) disks, 240
 - SATA (serial ATA) drives, 240–241
 - Scanner Access Now Easy (SANE)
 - project, 30
 - scanners for open ports, 532–535, 534–535
 - scanning log files, 172–177
 - scheduling
 - backups, 334–335
 - jobs, 125–129
 - scp command, 407
 - screen options in X configuration, 24–25
 - script kiddies, 545
 - scripted installations, 5
 - scripts, 89–91
 - log files, 168
 - starting and stopping services, 149–153, 152
 - Web servers, 490–492
 - SCSI (Small Computer System Interface)
 - interfaces, 239–241
 - searches
 - files, 63–66
 - log files, 174–175
 - man pages, 93–94
 - Vi, 136
 - second extended filesystem (ext2), 248–249
 - secondary boot loaders, 8
 - Secure Hash Algorithm (SHA), 223–224
 - secure log file, 171
 - Secure Shell (SSH) protocol
 - e-mail servers, 462
 - iptables traffic, 404
 - keys, 408–409
 - network backups, 334
 - remote access servers, 405–406
 - remote logins, 225–226
 - text-mode logins, 406–408
 - wireless networks, 515
 - Secure Sockets Layer (SSL) encryption
 - e-mail servers, 461–462
 - HTTPS, 519

security, 514

- auditing, 531–537, 534–535
- authentication. *See* authentication
- denial-of-service attacks, 518
- disabling accounts, 222–223
- encryption, 518–519
- exam essentials, 547
- FTP, 483, 519
- GPG, 526–529
- human element, 519
- intrusion detection. *See* intrusion detection
- local program bugs, 516–517
- NFS servers, 482–483
- partition options, 245
- passwords. *See* passwords
- physical access problems, 515
- review questions, 548–554
- root accounts, 192
- Samba, 479
- SELinux, 529–531
- server bugs, 517–518
- SMB/CIFS, 423–424
- stolen passwords, 516
- summary, 546
- super servers, 156–158
- threat categories, 514
- Security Enhanced Linux (SELinux), 529–531
- sed tool, 175–177
- See also section in man pages, 92
- SELinux (Security Enhanced Linux), 529–531
- semicolons (;)
 - commands, 70
 - /etc/dhcpd.conf, 440
 - /etc/named.conf, 442
 - log files, 165
- sendmail program
 - configuring, 457–458
 - queue management, 459

Sentry Tools package, 177

Sequenced Packet Exchange (SPX) protocol, 357

sequential-access devices, 324

serial ATA (SATA) drives, 240–241

Serial Attached SCSI (SAS) disks, 240

serial numbers in DNS records, 446–447

Server Message Block/Common Internet File System (SMB/CIFS), 271

in printing, 423–424

share access, 272–273

time setting, 449

server ports for firewalls, 396–398

ServerLayout section, 25

servers

access control, 224

bugs, 517–518

vs. clients, 365

e-mail, 461–462

name, 379–380

network. *See* network servers

remote, 169–170, 405–406

Samba setting, 479

super. *See* super servers

VNC, 496–497

X configuration, 17–20

Service Configuration tool, 152

services, 148

log files. *See* log files

starting and stopping, 148–149

custom startup files, 158–159

super servers, 153–158

SysV scripts, 149–153, 152

set command for physical volumes, 281

set group ID (SGID) bit, 78, 80, 517

set user ID (SUID) bit, 77–78, 80, 517

setenforce command, 531

setenv command, 86

setfacl command, 83–84

setup utility, 13

sftp program, 406–408

- sftp-server program, 406
- SGID (set group ID) bit, 78, 80, 517
- SGID program
 - finding, 124
 - risks, 123
 - uses, 123–124
- SHA (Secure Hash Algorithm), 223–224
- sha1sum program, 543
- shadow passwords
 - benefits, 221–222
 - user account support, 190
 - working with, 223–224
- sharing with Samba, 423, 475–478
- SHELL environment variable, 87
- shell scripts, 89–91
- shells, 30, 48–49
 - command history, 52–53
 - launching programs, 50–51
 - shortcuts, 51–52
 - starting, 49
 - user accounts, 205, 536
 - virtual terminals, 50
- short text files, viewing, 67–68
- shortcuts, 51–52
- shoulder surfing, 222
- SHOW DATABASES command, 502
- showmount command, 481–482
- shutdown command, 161–162
- signals, process, 117–118
- signing messages, 529
- Simple Mail Transfer Protocol (SMTP),
 - 381, 455
 - security issues, 519–520
 - server configuration, 457–461
- size, file
 - displaying, 73
 - log files, 168
- Slackware distribution, 3, 315
- slashes (/)
 - cron jobs, 126
 - directories, 55, 57, 59
 - /etc/named.conf, 442
 - log file searches, 175
 - netmasks, 360
 - root partitions, 245
 - sysctl keys, 131
- slocate program, 65
- slowdown as intrusion symptom, 537
- Small Computer System Interface (SCSI)
 - interfaces, 239–241
- smart filters, 416
- SMB/CIFS (Server Message Block/
 - Common Internet File System), 271
 - in printing, 423–424
 - share access, 272–273
 - time setting, 449
- smb.conf file, 475, 477–479
- smbclient program, 272
- smbd server, 475
- smbfs filesystem, 272
- smbmnt program, 272
- smbmount utility, 272
- smbpasswd command, 208, 476
- smbumount utility, 272
- SMTP (Simple Mail Transfer Protocol),
 - 381, 455
 - security issues, 519–520
 - server configuration, 457–461
- snort.conf.distrib file, 539
- Snort program, 538–540, 540
- SOA (start of authority) records, 446
- social engineering, 519
- sockets, 747
- soft links, 60–61
- software crashes as intrusion
 - symptom, 538
- solid-state storage, 239, 241
- source code compilation, 318–321
- source packages, 297
- source RPMs, 297
- spam, 457
- Special flag field, 206

- speed, RAID for, 276
- splash image option, 11
- SPX (Sequenced Packet Exchange)
 - protocol, 357
- SQL (Structured Query Language), 499–500
- SQLite package, 500
- square brackets ([])
 - ls command, 56
 - map pages, 92
 - regular expressions, 67
 - smb.conf file, 475, 477
- Squid proxy server, 494, 495
- SSH. *See* Secure Shell (SSH) protocol
- ssh command, 407
- ssh_host_dsa_key file, 408
- ssh_host_rsa_key file, 408
- ssh-keygen command, 409
- SSL (Secure Sockets Layer) encryption
 - e-mail servers, 461–462
 - HTTPS, 519
- stacks
 - module, 37
 - protocol, 354–358, 355
- standard error (stderr), 69
- standard filesystems, 274–276
- standard input (stdin), 68
- standard output (stdout), 67
- standards-based protocols, 356
- start command, 150
- start of authority (SOA) records, 446
- starting services, 148–149
 - custom startup files, 158–159
 - super servers, 153–158
 - SysV scripts, 149–153, 152
- startup files, 158–159
- startup issues
 - boot loaders, 7–13
 - boot problems, 14–17
 - exam essentials, 38–39
 - hardware, 30–37
 - installations, 2–7
 - review questions, 40–46
 - summary, 38
 - X configuration. *See* X Window System
- startup scripts, 27–28, 498
- startx command, 191, 411
- static IP address configuration, 369–373
- statistics, system, 119
 - general-purpose, 120–121
 - input/output, 121–122
 - memory, 122
- status
 - MySQL, 502
 - networks, 378–379
 - processes, 109–116, 114
- stderr (standard error), 69
- stdin (standard input), 68
- stdout (standard output), 67
- sticky bit, 78, 80
- stolen passwords, 516
- stop command, 150
- stopping MySQL, 501
- stopping services, 148–149
 - custom startup files, 158–159
 - super servers, 153–158
 - SysV scripts, 149–153, 152
- storage hardware, 238
 - configuration, 240–242
 - device types, 239–240
 - disks. *See* disks
- stratum time servers, 449
- striping, RAID, 277
- strong passwords, 221
- Structured Query Language (SQL), 499–500
- su program, 124, 191
- subdomains, 362
- subnets
 - DHCP, 440
 - masks, 360

- sudo command, 192
- SUID (set user ID) bit, 77, 79, 517
- SUID program
 - finding, 124
 - risks, 123
 - uses, 123–124
- SUID root files, 192
- summary searches in man pages, 93–94
- super servers, 148–149
 - inted.conf and inetd.d files, 153–155
 - security, 156–158
 - starting and stopping services, 153–158
 - xinted.conf and xinetd.d files, 155–156
- superusers, 191–192
- support libraries, 318–319
- SUSE distribution, 3
- suspicious logins, 545–546
- swap space
 - adding, 261–262
 - evaluating, 260–261
 - overview, 259
 - partitions, 246, 251, 262–263
- swapon command, 261–262
- swapon command, 261
- SWAT (Samba Web Administration Tool), 226
- switches, 352, 353
- symbolic links, 60–61
 - file type codes, 75
 - optical disks, 338
 - permissions, 77
- symbolic mode, 80–81
- synchronizing computers, 333–334
- Synopsis section in man pages, 92
- sysctl utility, 131–133, 393
- syslog file, 171
- syslog-ng system logger, 163
- syslogd daemon, 163–164, 169
- syslogd package, 163
- system account passwords, 536
- system-config-network-gui tool, 373–374

- system-config-network-tui tool, 373
- system control data, 131–133
- system crashes as intrusion
 - symptom, 538
- system cron jobs, 125–127
- system resources in compilation, 319
- system services. *See* services
- system slowdown as intrusion
 - symptom, 537
- system statistics, 119
 - general-purpose, 120–121
 - input/output, 121–122
 - memory, 122
- SystemRescueCd package, 15
- SysV scripts
 - starting and stopping services, 149–153, 152
 - startup, 27–28

T

- tables
 - iptables, 398–404, 399
 - routing, 531
- tail program, 68, 173–174
- tape drives
 - for creating backups, 325–326
 - for restoring backups, 331–333
- tar program
 - for backups, 328–329
 - overview, 314–317
- tarballs, 295, 314
 - creating, 318
 - role, 314–315
 - tar, 316–317
- TCP (Transmission Control Protocol), 356
- TCP/IP (Transmission Control Protocol/Internet Protocol) stack, 356
- TCP Wrappers, 156–157
- tcpd command, 156

- tcsh shell, 49, 86
- teatime setting, 128
- tee command, 69
- teletype (TTY) code, 112
- telinit program, 15, 27, 160–162
- telnet package, 405
- telnet program
 - as diagnostic tool, 381
 - port numbers, 382
 - remote access servers, 405
 - security issues, 519
 - servers, 225–226
- telnetd server, 405
- 10GBase networks, 351
- TERM environment variable, 88
- terminal programs, 29–30
- terms, glossary, 556–591
- test command, 78
- testing
 - connectivity, 377
 - MySQL connections, 501–502
 - recovery tools, 336
- text-based installers, 5
- text-based Web clients, 495–496
- text files
 - combining, 67–68
 - editing, 134–136, 135
 - log files, 173
 - viewing, 67–68
- text-mode commands, 48
 - environment variables, 84–89
 - exam essentials, 98
 - files and directories. *See* directories; files
 - help resources, 91–97
 - logins, 406–408
 - review questions, 99–105
 - shell scripts, 89–91
 - shells, 48–53
 - summary, 97
 - Vi, 134
- third extended filesystem (ext3fs), 249–250
- thorough searches in man pages, 94
- Tight VNC site, 498–499
- tightvnc package, 498
- tightvnc-server package, 498
- tildes (~)
 - backup files, 70
 - home directories, 57, 489
 - Vi, 134
- time
 - file creation, 74
 - NTP, 448–454, 450, 453
- time setting
 - cron jobs, 126
 - installation process, 6
 - job scheduling, 128
 - log files, 168
- time-to-live (TTL) value, 446
- Timeout period option in GRUB, 11
- timestamps, 60
- TLDs (top-level domains), 362, 441
- /tmp partition, 247
- Token Ring networks, 351
- Tomcat scripts, 492
- top command, 113–116, 114
- top-level domains (TLDs), 362, 441
- Top page command for info pages, 95
- touch command, 60
- tpb utility, 242
- traceroute command, 377–378
- transfer modes in FTP, 488
- Transmission Control Protocol (TCP), 356
- Transmission Control Protocol/Internet Protocol (TCP/IP) stack, 356
- transposing text, 53
- Triple Data Encryption Standard (3DES) hash, 223
- Tripwire utility, 542–543
- True-Crypt program, 515
- tshark command, 541
- TTL (time-to-live) value, 446

tvwm window manager, 29
 tw.cfg file, 543
 tw.pol file, 543
 twcfg.txt file, 543
 twinstall.sh program, 543
 two-factor authentication, 526
 twpol.txt file, 543
 TXT records, 446
 type codes for files, 74–78
 type information for files, 78

U

Ubuntu distribution
 description, 3
 display manager setting, 27
 runlevels, 160
 Samba server, 150
 static IP addresses, 370–371
 UDF (Universal Disk Format), 249, 339
 UDP (User Datagram Protocol), 356
 UFS (Unix Filesystem), 249
 UIDs (user IDs)
 mapping to users, 194–196
 root, 536
 user accounts, 72–73, 189, 205
 umask command, 82–83
 umount command, 266, 270–271
 uname command, 129–131
 underscores (_) in usernames, 193
 undo command in Vi, 136
 uniform resource identifiers (URIs), 423
 Universal Coordinated Time (UTC), 6
 Universal Disk Format (UDF), 249, 339
 Universal Serial Bus (USB) devices, 32–33
 Unix Filesystem (UFS), 249
 Unix98 ps options, 110–111
 unknown accounts, 536
 unmounting partitions, 270–271
 unprivileged ports, 398
 upgrading
 kernels, 303
 packages, 323
 Upstart package, 163
 URIs (uniform resource identifiers), 423
 USB (Universal Serial Bus) devices, 32–33
 user accounts. *See* users and user accounts
 user complaints as intrusion
 symptom, 538
 user cron jobs, 125, 127–128
 User Datagram Protocol (UDP), 356
 USER environment variable, 87
 user IDs (UIDs)
 mapping to users, 194–196
 root, 536
 user accounts, 72–73, 189, 205
 user masks, 82–83
 user-mode processes, 129
 user private groups, 217
 useradd utility, 197–200
 userdel command, 207–208, 222
 usermod program
 groups, 214
 passwords, 202
 usernames
 processes, 112
 user accounts, 189, 193, 204–205
 users and user accounts, 188
 for access control, 224–226
 adding, 197–200
 authentication. *See* authentication
 configuration files, 204–207
 deleting, 207–208
 disabling unused accounts, 222–223
 exam essentials, 227–228
 FTP configuration, 485
 groups. *See* groups
 home directories, 196–197
 in installation process, 6
 mapping UIDs to, 194–196
 multitasking systems, 190–191

- multiuser systems, 189–190
- passwords, 200–204, 219–224
- permissions, 72, 189
- review questions, 229–235
- reviewing, 535–536
- security, 219–224
- summary, 227
- superusers, 191–192
- usernames, 193
- verifying account use, 208–212
- /usr/doc directory, 97
- /usr partition, 246
- /usr/local partition, 246
- /usr/share/doc directory, 97
- UTC (Universal Coordinated Time), 6
- UW IMAP servers, 461

V

- /var/lib/dpkg directory, 295
- /var/lib/ntp directory, 450
- /var/lib/rpm directory, 295
- /var/log directory, 16, 165
- /var/log/dmesg file, 173
- /var/log/kernel file, 165
- /var/log/mail file, 165
- /var/log/messages file, 150, 174
- /var/log/snort file, 539
- /var partition, 247
- /var/spool/cron directory, 125
- /var/spool/cups directory, 414
- verifying
 - account use, 208–212
 - identity, 212
 - installed files and packages, 537
 - messages in GPG, 529
- versions
 - kernels, 129–131
 - packages, 324
 - RPM, 299
- vertical bars (|) for pipes, 69
- VertRefresh setting, 23
- VFAT (Virtual FAT), 249
- vgcreate command, 281
- vgdisplay command, 282
- vgremove command, 282
- vgrename command, 282
- vgscan command, 282
- vgsplit command, 282
- Vi editor, 132
 - modes, 132–133
 - saving changes, 137
 - text editing, 134–136, 135
- video cards
 - chipsets, 18–19, 18
 - options, 23–24
- viewing files, 67–68
- Virtual FAT (VFAT), 249
- virtual hosting, 492–493
- Virtual Network Computing (VNC),
 - 411, 496
 - clients, 498–499
 - server configuration, 496–497
- virtual terminals (VTs), 7,
 - 50, 191
- visudo command, 192
- vmlinux kernel, 13
- vmlinux kernel, 13
- vmstat utility, 122
- VNC (Virtual Network Computing),
 - 411, 496
 - clients, 498–499
 - server configuration, 496–497
- vnc package, 498
- VNC program, 191
- vnc-server package, 498
- vncpasswd program, 498
- vncserver command, 498
- vncviewer program, 498–499
- volume groups, 280–282

vsftpd FTP servers, 484–486
 VTs (virtual terminals), 7, 50, 191

W

w command, 208–210
 WAPs (wireless access points), 353, 367, 367
 warnings for log files, 164–166
 Washington University FTP Daemon (WU-FTPD) FTP servers, 484
 Web-based CUPS utilities, 420–422, 421
 Web servers
 Apache, 489–490, 493–494
 proxy, 494–495, 495
 scripts, 490–492
 text-based clients, 495–496
 virtual hosting, 492–493
 weekly option for log files, 168
 Weinberger, Peter J., 175
 WEP (Wired Equivalent Privacy), 367, 515
 wget program, 495–496
 whatis command, 94
 whereis command, 65–66
 who command, 208–210
 whoami command, 208, 212
 Wi-Fi Protected Access 2 (WPA2), 367, 515
 wildcard characters, 56
 Winbind server, 455, 479
 winbindd tool, 455, 479
 window managers, 29
 Windows
 remote access tools, 496
 rdesktop, 496–497
 VNC, 497–499
 with Samba, 478–479
 video drivers, 18

Wired Equivalent Privacy (WEP), 367, 515
 wireless access points (WAPs), 353, 367, 367
 wireless networks
 options, 366–368, 367
 protocols, 352
 security, 515
 Wireshark packet sniffer, 541–542, 542
 workgroup parameter in Samba, 475
 world permissions, 75
 WPA2 (Wi-Fi Protected Access 2), 367, 515
 Wrappers, TCP, 156–157
 write permission, 75–77
 WU-FTPD (Washington University FTP Daemon) FTP servers, 484

X

X-CD-Roast tool, 337
 X Display Manager (XDM), 26
 configuring, 28
 network connections, 411
 X Display Manager Control Protocol (XDMCP), 26
 configuring, 412–413
 servers, 26, 26
 configuring, 28–29
 running, 27–28
 VNC linked to, 497
 X.org-X11 servers, 19
 X Window System, 2
 clients, 17
 configuration, 17
 GUI logins, 26, 26
 installation process, 6
 methods, 21–22
 miscellaneous options, 22
 monitor options, 23

- screen options, 24–25
- server selection, 17–20
- setup, 20
- terminal programs, 29–30
- video card chipsets, 18–19, 18
- video card options, 23–24
- window managers and desktop environments, 29
- XDMCP server, 27–29
- port forwarding, 406
- programs, 20, 409–411
- servers, 17, 21, 413
- terminals, 413
- xargs command, 70
- Xconfigurator tool, 21
- XDM (X Display Manager), 26
 - configuring, 28
 - network connections, 411
- xdm script, 27
- XDMCP (X Display Manager Control Protocol), 26
 - configuring, 412–413
 - servers, 26, 26
 - configuring, 28–29
 - running, 27–28
 - VNC linked to, 497
- XF86Config file, 20
- xf86config tool, 21
- XF86Setup tool, 21
- Xfce desktop environment, 29
- XFree86 server, 19–22
- XFree86-server package, 20
- XFS (Extent Filesystem), 249–250
- xfsdump program, 329
- xfsrestore program, 329
- xhost command, 410
- Xi Graphics company, 20
- xinted.conf file, 155–156
- xinetd.d directory, 155–156

- xinetd program, 156–158, 405
- XkbLayout option, 22
- XkbModel option, 22
- xntp package, 452
- xntpd package, 452
- Xorg.0.log file, 171
- xorg.conf file, 21–22
- Xorg program, 20–21, 413
- xserver-xfree86 package, 20
- xterm program, 29–30, 49

Y

- Yaboot boot loader, 245
- YaST tool, 152
- YaST2 tool, 517
- Yellow Pages (YP) protocol, 523
- yp-tools package, 523
- ypbind daemon, 524
- ypbind package, 523
- yppasswd command, 524
- yum command, 304–305
- Yum Extender (yumex) tool, 306
- Yum meta-packager, 303–307, 306
- Yum tool, 517
- yumdownloader command, 305
- yumex (Yum Extender) tool, 306

Z

- zImage kernel, 13
- zip files, 315
- zones
 - BIND for, 443–448
 - forward, 444–447
 - reverse, 447–448
 - root, 443

Wiley Publishing, Inc. End-User License Agreement

READ THIS. You should carefully read these terms and conditions before opening the software packet(s) included with this book “Book”. This is a license agreement “Agreement” between you and Wiley Publishing, Inc. “WPI”. By opening the accompanying software packet(s), you acknowledge that you have read and accept the following terms and conditions. If you do not agree and do not want to be bound by such terms and conditions, promptly return the Book and the unopened software packet(s) to the place you obtained them for a full refund.

1. License Grant. WPI grants to you (either an individual or entity) a nonexclusive license to use one copy of the enclosed software program(s) (collectively, the “Software”) solely for your own personal or business purposes on a single computer (whether a standard computer or a workstation component of a multi-user network). The Software is in use on a computer when it is loaded into temporary memory (RAM) or installed into permanent memory (hard disk, CD-ROM, or other storage device). WPI reserves all rights not expressly granted herein.

2. Ownership. WPI is the owner of all right, title, and interest, including copyright, in and to the compilation of the Software recorded on the physical packet included with this Book “Software Media”. Copyright to the individual programs recorded on the Software Media is owned by the author or other authorized copyright owner of each program. Ownership of the Software and all proprietary rights relating thereto remain with WPI and its licensors.

3. Restrictions on Use and Transfer.

(a) You may only (i) make one copy of the Software for backup or archival purposes, or (ii) transfer the Software to a single hard disk, provided that you keep the original for backup or archival purposes. You may not (i) rent or lease the Software, (ii) copy or reproduce the Software through a LAN or other network system or through any computer subscriber system or bulletin-board system, or (iii) modify, adapt, or create derivative works based on the Software.

(b) You may not reverse engineer, decompile, or disassemble the Software. You may transfer the Software and user documentation on a permanent basis, provided that the transferee agrees to accept the terms and conditions of this Agreement and you retain no copies. If the Software is an update or has been updated, any transfer must include the most recent update and all prior versions.

4. Restrictions on Use of Individual Programs. You must follow the individual requirements and restrictions detailed for each individual program in the “About the CD” appendix of this Book or on the Software Media. These limitations are also contained in the individual license agreements recorded on the Software Media. These limitations may include a requirement that after using the program for a specified period of time, the user must pay a registration fee or discontinue use. By opening the Software packet(s), you agree to abide by the licenses and restrictions for these individual programs that are detailed in the “About the CD” appendix and/or on the Software Media. None of the material on this Software Media or listed in this Book may ever be redistributed, in original or modified form, for commercial purposes.

5. Limited Warranty.

(a) WPI warrants that the Software and Software Media are free from defects in materials and workmanship under normal use for a period of sixty (60) days from the date of purchase of this Book. If WPI receives notification within

the warranty period of defects in materials or workmanship, WPI will replace the defective Software Media.

(b) WPI AND THE AUTHOR(S) OF THE BOOK DISCLAIM ALL OTHER WARRANTIES, EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, WITH RESPECT TO THE SOFTWARE, THE PROGRAMS, THE SOURCE CODE CONTAINED THEREIN, AND/OR THE TECHNIQUES DESCRIBED IN THIS BOOK. WPI DOES NOT WARRANT THAT THE FUNCTIONS CONTAINED IN THE SOFTWARE WILL MEET YOUR REQUIREMENTS OR THAT THE OPERATION OF THE SOFTWARE WILL BE ERROR FREE.

(c) This limited warranty gives you specific legal rights, and you may have other rights that vary from jurisdiction to jurisdiction.

6. Remedies.

(a) WPI's entire liability and your exclusive remedy for defects in materials and workmanship shall be limited to replacement of the Software Media, which may be returned to WPI with a copy of your receipt at the following address: Software Media Fulfillment Department, Attn.: *CompTIA Linux+ Study Guide*, Wiley Publishing, Inc., 10475 Crosspoint Blvd., Indianapolis, IN 46256, or call 1-800-762-2974. Please allow four to six weeks for delivery. This Limited Warranty is void if failure of the Software Media has resulted from accident, abuse, or misapplication. Any replacement Software Media will be warranted for the remainder of the original warranty period or thirty (30) days, whichever is longer.

(b) In no event shall WPI or the author be liable for any damages whatsoever (including without limitation damages for loss of business profits, business interruption, loss of business information, or any other pecuniary loss) arising from the use of or inability to use the Book or the Software, even if WPI has been advised of the possibility of such damages.

(c) Because some jurisdictions do not allow the exclusion or limitation of liability for consequential or incidental damages, the above limitation or exclusion may not apply to you.

7. U.S. Government Restricted Rights. Use, duplication, or disclosure of the Software for or on behalf of the United States of America, its agencies and/or instrumentalities “U.S. Government” is subject to restrictions as stated in paragraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause of DFARS 252.227-7013, or subparagraphs (c) (1) and (2) of the Commercial Computer Software - Restricted Rights clause at FAR 52.227-19, and in similar clauses in the NASA FAR supplement, as applicable.

8. General. This Agreement constitutes the entire understanding of the parties and revokes and supersedes all prior agreements, oral or written, between them and may not be modified or amended except in a writing signed by both parties hereto that specifically refers to this Agreement. This Agreement shall take precedence over any other documents that may be in conflict herewith. If any one or more provisions contained in this Agreement are held by any court or tribunal to be invalid, illegal, or otherwise unenforceable, each and every other provision shall remain in full force and effect.

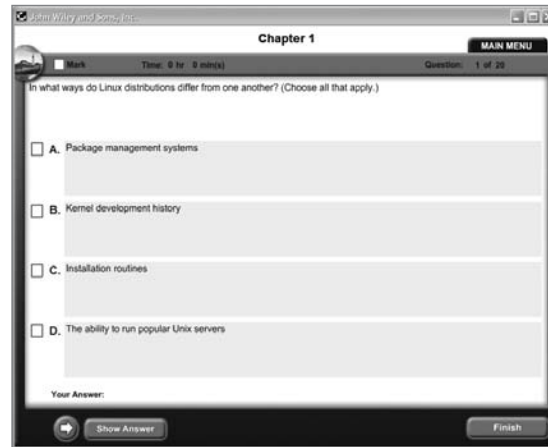
The Best Linux+ Book/CD Package on the Market!



Get ready for your Linux+ certification with the most comprehensive and challenging sample tests anywhere!

The Sybex Test Engine features:

- All the review questions, as covered in each chapter of the book.
- Challenging questions representative of those you'll find on the real exam.
- Two full-length bonus exams available only on the CD.
- An Assessment Test to narrow your focus to certain objective groups.



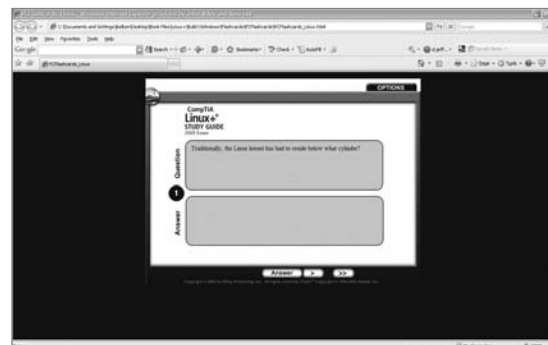
Search through the complete book in PDF!

- Access the entire *CompTIA Linux+ Study Guide* complete with figures and tables, in electronic format.
- Search the *CompTIA Linux+ Study Guide* chapters to find information on any topic in seconds.



Use the Electronic Flashcards to jog your memory and prep last-minute for the exam!

- Reinforce your understanding of key concepts with these hardcore flashcard-style questions.



CompTIA Linux+ Study Guide

Exam XK0-003

OBJECTIVE	CHAPTER
1.0 Installation and Configuration	
1.1 Compare and contrast installation sources Physical installation media: CD-ROM, DVD Network types: HTTP, FTP, NFS	1
1.2 Implement partitioning schemes and filesystem layout using the following tools and practices LVM, RAID, fdisk, parted, mkfs	6
1.3 Explain the purpose for using each of the following filesystem types Local: EXT2, EXT3, Reiser, FAT, NTFS, VFAT, ISO9660 Network: NFS, SMBFS / CIFS	6
1.4 Conduct routine mount and unmount of filesystems mount, umount, /etc/fstab	6
1.5 Explain the advantages of having a separate partition or volume for any of the following directories /boot, /home, /tmp, /usr, /var, /opt	6
1.6 Explain the purpose of the following directories /, /bin, /dev, /etc, /mnt, /proc, /root, /sbin, /usr/bin, /usr/local, /usr/lib, /usr/lib64, /usr/share, /var/log	6
1.7 Configure the boot process including the following GRUB: /boot/grub/grub.conf, /boot/grub/menu.lst, grub-install, grub	1
1.8 Perform the following package management functions Install, remove and update programs: rpm (rpm -Uvh, rpm -qa, rpm -e, yum), deb (dpkg -i, dpkg -r, apt-get, apt-cache search), source (./configure, make, make install, make uninstall, tar, make clean, autoconf, make test, tar.gz, INSTALL, bzip, gzip) Resolve dependencies Add and remove repositories	7
1.9 Configure profile and environment variables system-wide and at the user level PS1, PS2, PATH, EDITOR, TERM, PAGER, HOME, PRINTER	2
1.10 Troubleshoot boot issues using the following tools Kernel options, Single-user mode (including recovering the root user), Rescue — live CDs, DVDs and USB keys, dmesg	1
1.11 Manage devices using the following tools lsusb, lspci, lsmod, /sys, modprobe, /proc, /etc/modules.conf, /etc/modprobe.conf, Hardware Compatibility List (HCL)	1
2.0 System Maintenance and Operations	
2.1 Given a scenario, use the following fundamental Linux tools, techniques and resources	2, 3

OBJECTIVE	CHAPTER
<p>Directory navigation (cd, ls, pushd, popd, pwd)</p> <p>File commands: file, test, find, locate, slocate, which, whereis, ln, ls -F, mknod, touch, mkdir, mv, cp, rm, cd, file types (hardlinks, softlinks, directory, device file, regular file, named pipe)</p> <p>File editing with vi</p> <p>Process management: ps, kill, top, iostat, pstree, nice, renice, signals, PID, PPID</p> <p>I/O redirection: <, >, =, ==, , ;, tee, xargs, STDIN, STDOUT, STDERR</p> <p>Special devices: /dev/null, /dev/random, /dev/zero, /dev/urandom</p> <p>System documentation: Man pages(man #, apropos, makewhatis, whatis), Info pages, /usr/share/docs</p> <p>Virtual consoles</p> <p>Kernel / architecture information: cat, /proc/version, uname, common sysctl settings, /etc/sysctl.conf</p>	
2.2 Conduct basic tasks using BASH	2
<p>Basics of scripting (only: execute permission, #!/bin/bash, sh script)</p> <p>Shell features: history, tab completion</p>	
2.3 Given a scenario, analyze system and application logs to troubleshoot Linux systems	4
<p>Common log files: /var/log/messages, /var/log/syslog, /var/log/maillog, /var/log/secure, /var/log/lastlog</p> <p>Rotated logs</p> <p>Searching and interpreting log files: grep, tail -f, awk, sed</p>	
2.4 Conduct and manage backup and restore operations	7
<p>Copying data: rsync and ftp</p> <p>Archive and restore commands: cpio, tar, dump, restore, dd</p> <p>Write to removable media (CD-RW, DVD-RW)</p>	
2.5 Explain the following features and concepts of X11	1
<p>Starting and stopping X11, Difference between the X11 client and server, Window managers and display managers (KDM, GDM), Multiple desktops, X11 configuration file (xorg.conf), Terminal emulators (xterm, etc)</p>	
2.6 Explain the difference in runlevels and their purpose	4
<p>Command: init</p> <p>Runlevels: 0 – Halt, 1 – single-user mode, 2 – single-user mode with networking, 3 – networked multi-user mode, 4 – user configurable, 5 – X11 multi-user mode, 6 – reboot</p>	
2.7 Manage filesystems using the following	6, 11
<p>Check disk usage (df, du)</p> <p>Quotas: edquota, repquota, quotacheck</p> <p>Check and repair filesystems (fsck)</p> <p>Loopback devices (ISO filesystems)</p> <p>NFS: configuration, mount, exportfs, fstab, /etc/exports, showmount</p> <p>Swap: mkswap, swapon, swaponoff</p>	

OBJECTIVE	CHAPTER
2.8 Implement task scheduling using the following tools cron (cron.allow, cron.deny), crontab command syntax, crontab file format, at (atq)	3
2.9 Utilize performance monitoring tools and concepts to identify common problems Commands: sar, iostat, vmstat, uptime, top Load average	3
3.0 Application and Services	
3.1 Manage Linux system services using the following /etc/init.d (start, stop, restart) inetd, xinetd, chkconfig	4
3.2 Implement interoperability with Windows using the following rdesktop – client vnc – server and client Samba – server and client: smb.conf, Winbind, lmhosts Security and authentication (Kerberos)	10, 11
3.3 Implement, configure and maintain Web and FTP services Apache: Maintain PHP settings (php.ini), Edit Apache configuration files (Enable and disable modules), Containers (Virtual hosts, Directories), Access control (.htaccess), CGI (ExecCGI, ScriptAlias), Commands: apachectl (-t, -S, graceful, restart), Configuring apache logs FTP services: Configure FTP users (/etc/ftpusers, chroot), Configure anonymous access	11
3.4 Given a scenario, explain the purpose of the following web-related services Tomcat, Apache, Squid	11
3.5 Troubleshoot web-related services using the following utilities Commands: curl, wget, ftp, telnet	11
3.6 Given a scenario, troubleshoot common FTP problems Active vs. passive, ASCII vs. binary	11
3.7 Given a scenario, perform the following MySQL administrative tasks Locate configuration file, Starting and stopping, Test the connection	11
3.8 Explain the purpose of each of the following mail services, protocols and features Protocols: SMTP, IMAP, POP3 MTA: Postfix, Sendmail Email aliases: /etc/aliases, newaliases	10
3.9 Deploy and manage CUPS print services Enable and disable queues Web management interface (port 631) Printing commands: lpr, lp, lpq, lpstat, cancel	9
3.10 Set up, install, configure and maintain a BIND DNS server and related services DNS utilities: named, rndc Config file locations (/var/named) Forward zones, reverse zones, root hints	10

OBJECTIVE	CHAPTER
3.11 Perform basic administration of the DHCP server /etc/dhcpd.conf, dhcpd.leases	10
3.12 Given a scenario, troubleshoot NTP related issues /etc/ntp.conf, ntpdate, date, ntpq -p	10
4.0 Networking	
4.1 Identify common networking ports and the associated service 20, 21, 22, 23, 25, 53, 80, 110, 123, 143, 443, 631, 3306, /etc/services	9
4.2 Execute network interface configuration using the following dhclient, dhcpd, ifconfig, iwconfig, route, ifup, ifdown, network configuration files	8
4.3 Implement configurations and/or configuration changes for the following Packet filtering: iptables Hostname lookup: /etc/hosts, /etc/nsswitch.conf, /etc/resolv.conf	8, 9
4.4 Explain the different DNS record types and the process of DNS resolution Local resolution, TTL/caching, Root name servers, A, MX, PTR, CNAME, NS, TXT	10
4.5 Troubleshoot basic connectivity issues using the following tools netstat, ping, traceroute, arp, telnet, route	8
4.6 Troubleshoot name resolution issues using the following tools dig, host, nslookup, hostname	8
5.0 Security	
5.1 Manage and monitor user and group accounts using the following Tools: useradd, userdel, usermod, groupadd, groupdel, groupmod, lock, who, w, last, whoami Files: /etc/skel, /etc/passwd, /etc/shadow, /etc/group	5
5.2 Given a scenario, select the appropriate file permissions and ownership and troubleshoot common problems Tools: chmod, chown, chroot, chgrp, lsattr, chattr, umask Special permissions: setuid, setgid, sticky bit	2
5.3 Explain the basics of SELinux Running modes: Enabled, Disabled, Permissive	12
5.4 Given a scenario, implement privilege escalation using the following sudo, su, /etc/sudoers	5
5.5 Explain the appropriate use of the following security related utilities nmap, Wireshark, NESSUS, Snort, Tripwire	12
5.6 Use checksum and file verification utilities md5sum, sha1sum, gpg	12
5.7 Deploy remote access facilities using the following SSH: Secure tunnels, SFTP, X11 forwarding, Keygen VNC	9
5.8 Explain the methods of authentication PAM, LDAP, NIS, RADIUS, Two-factor authentication	12



Exam objectives are subject to change at any time without prior notice and at CompTIA's sole discretion. Please visit CompTIA's website (www.comptia.org) for the most current listing of exam objectives.

Prepare for the new 2009 Linux+ exam

As the Linux server and desktop markets continue to grow, so does the need for qualified Linux administrators. CompTIA's new Linux+ (Exam XK0-003) includes the very latest enhancements to the popular open source operating system. This detailed guide not only covers all key exam topics—including installation and configuration, system maintenance and operations, application and services, networking and security—it also builds your practical Linux skills with real-world examples. Inside, you'll find:

- Full coverage of all exam objectives** in a systematic approach, so you can be confident you're getting the instruction you need for the exam
- Real-world scenarios** that put what you've learned into practical context
- Challenging review questions** in each chapter to prepare you for exam day
- Exam Essentials**, a key feature in each chapter that identifies critical areas you must become proficient in before taking the exam
- A handy tear card** that maps every official exam objective to the corresponding chapter in the book, so you can track your exam prep objective by objective

Look inside for complete coverage of all exam objectives.

www.sybex.com

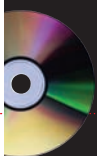
ABOUT THE AUTHOR


Roderick W. Smith, Linux+, LPIC-1, is a Linux consultant and author. His areas of expertise include Linux networking, filesystems, and cross-platform configuration. He has written over 20 books on open source technologies, including the *LPIC-1: Linux Professional Institute Certification Study Guide, 2nd Edition* and *Linux Administrator Street Smarts*, both from Sybex.


Sybex®
An Imprint of
WILEY



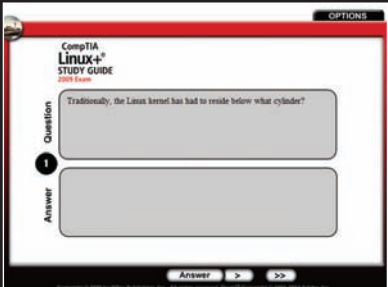
\$49.99 US
\$59.99 CN

**FEATURED ON THE CD**

**STUDY GUIDE**

**CompTIA
Linux+
STUDY GUIDE**

SYBEX TEST ENGINE:
Test your knowledge with advanced testing software. Includes all chapter review questions and bonus exams. Runs on both Windows and Linux.

**ELECTRONIC FLASHCARDS:**
Reinforce your understanding with electronic flashcards.


Also on CD, you'll find the entire book in searchable and printable PDF. Study anywhere, any time, and approach the exam with confidence.


CATEGORY
COMPUTERS/Certification Guides

Smith

Exam XK0-003

ISBN: 978-0-470-50384-3





Covers All Exam Objectives



Includes Real-World Scenarios and Leading-Edge Exam Prep Software Featuring:

- Linux-Compatible Custom Test Engine
- Hundreds of Sample Questions
- Electronic Flashcards
- Entire Book in PDF



CompTIA Linux+ STUDY GUIDE

Exam XK0-003

Roderick W. Smith



SERIOUS SKILLS.